

Biabduction (and Related Problems) in Array Separation Logic

James Brotherston

University College London, UK
J.Brotherston@ucl.ac.uk

Nikos Gorogiannis

Middlesex University, UK
nikos.gorogiannis@gmail.com

Max Kanovich

University College London, UK and
National Research University Higher
School of Economics, Russian
Federation
M.Kanovich@ucl.ac.uk

Abstract

We investigate *array separation logic* (ASL), a variant of symbolic-heap separation logic in which the data structures are either pointers or *arrays*, i.e., contiguous blocks of allocated memory. This logic provides a language for compositional memory safety proofs of imperative array programs.

We focus on the *biabduction* problem for this logic, which has been established as the key to automatic specification inference at the industrial scale. We present an NP decision procedure for biabduction in ASL that produces solutions of reasonable quality, and we also show that the problem of finding a consistent solution is NP-hard.

Along the way, we study satisfiability and entailment in our logic, giving decision procedures and complexity bounds for both problems. We show satisfiability to be NP-complete, and entailment to be decidable with high complexity. The somewhat surprising fact that biabduction is much simpler than entailment is explained by the fact that, as we show, the element of choice over biabduction solutions enables us to dramatically reduce the search space.

Categories and Subject Descriptors F.3.1 [Specifying and Verifying and Reasoning about Programs]: Logics of programs; D.2.4 [Software/Program Verification]: Assertion checkers; F.2 [Analysis of Algorithms and Problem Complexity]

Keywords Separation logic, arrays, biabduction, satisfiability, entailment, complexity.

1. Introduction

In the last 15 years, *separation logic* [33] has evolved from a novel way to reason about memory pointers to a mainstream

technique for scalable program verification. Facebook’s INFER [12] static analyser is perhaps the best known tool based on separation logic; other examples include SLAYER [5], VERIFAST [26] and the HIP tool series [13].

Separation logic is based upon *Hoare triples* of the form $\{A\} C \{B\}$, where C is a program and A, B are formulas in a logical language. Its compositional nature, the key to scalable analysis, is supported by two main pillars. The first pillar is the soundness of the following *frame rule*:

$$\frac{\{A\} C \{B\}}{\{A * F\} C \{B * F\}} \text{ (Frame)}$$

where the *separating conjunction* $*$ is read, intuitively, as “*and separately in memory*”, and subject to the restriction that C does not modify any free variables in F [39].

The second pillar is a tractable algorithm for the *biabduction* problem [11]: given formulas A and B , find “antiframe” and “frame” formulas X, Y respectively with

$$A * X \models B * Y,$$

usually subject to the proviso that $A * X$ should be satisfiable. Solving this problem enables us to infer specifications for whole programs given specifications for their individual components [11]. E.g., if C_1 and C_2 have specifications $\{A'\} C_1 \{A\}$ and $\{B\} C_2 \{B'\}$, we can use a solution X, Y to the above biabduction problem to construct a specification for $C_1; C_2$ as follows, using the frame rule and the usual Hoare logic rules for consequence (\models) and sequencing ($;$):

$$\frac{\frac{\frac{\{A'\} C_1 \{A\}}{\{A' * X\} C_1 \{A * X\}} \text{ (Frame)}}{\{A' * X\} C_1 \{B * Y\}} \text{ (}\models\text{)}}{\{A' * X\} C_1; C_2 \{B' * Y\}} \text{ (Frame)}$$

Bottom-up interprocedural analyses based on separation logic, such as Facebook INFER, employ biabduction in this way to infer program specifications bottom-up from unannotated code. Typically, the underlying language of assertion formulas is based on the “symbolic heap” fragment of

separation logic over linked lists [4], which is known to be tractable [14].

In this paper, we instead focus on a different, but similarly ubiquitous data structure for imperative programming, namely *arrays*, which we view as contiguous blocks of allocated heap memory. We propose an *array separation logic* (ASL) in which we replace the usual “list segment” predicate ls of separation logic by an “array” predicate $\text{array}(a, b)$, which denotes a contiguous block of allocated heap memory from address a to address b (inclusive), as was first proposed in [31]. In addition, since we wish to reason about array bounds, we also allow our assertions to contain linear arithmetic. Thus, for example, a pointer x to an memory block of length $n > 1$ and starting at a can be represented in ASL by the assertion

$$n > 1 : x \mapsto a * \text{array}(a, a + n - 1) .$$

The array predicate only records the bounds of memory blocks, and not their contents; this is analogous to the abstraction from pointers to lists in standard separation logic. Indeed, the memory safety of array-manipulating programs typically depends only on the memory footprint of the arrays. E.g., the usual quicksort and mergesort procedures for arrays work by partitioning the array and recursing based on a pivot chosen from among the allocated addresses.

Our focus in this paper is on the biabduction problem, as above, for ASL. Solving this problem is, we believe, *the* most critical step in building a bottom-up memory safety analysis à la INFER for array-manipulating programs. The first main contribution of the current work is a decision procedure for the (quantifier-free) biabduction problem in ASL, which we present in Section 5. It relies on the idea that, given A and B , we can look for some consistent total ordering of all the array endpoints and pointer addresses in both A and B , and impose this ordering, which we call a *solution seed*, as the arithmetical part of the solution X . Having done this, the computation of the “missing” arrays and pointers in X and Y becomes a polynomial-time process, and thus the entire algorithm runs in NP-time. We demonstrate that, as well as being sound, this algorithm is in fact complete; a biabduction solution exists if and only if a solution seed exists. We also show that the biabduction problem is NP-hard, and give further bounds for cases involving quantifiers.

Along the way, we study the *satisfiability* and *entailment* problems in ASL, and, as our second main contribution, we provide decision procedures and upper/lower complexity bounds for both problems. We find that satisfiability is NP-complete, while entailment is decidable with very high complexity: it can be encoded in Π_2^0 Presburger arithmetic, and is also at least Π_2^P -hard. The fact that entailment is much harder than biabduction may at first sight appear surprising, since biabduction also seems to involve solving an entailment problem. However, in the biabduction problem, there is an element of *choice* over X and Y , and we can exploit

this in a way that dramatically reduces the cost of checking these conditions. Namely, committing to a specific solution seed (see above) reduces biabduction to a simple computation rather than a search problem.

The remainder of this paper is structured as follows. Section 2 gives some examples showing how the ASL biabduction problem arises in verification practice; the syntax and semantics of ASL is then presented formally in Section 3. We present algorithms and establish complexity bounds for satisfiability, biabduction and entailment for ASL in Sections 4, 5 and 6 respectively. Section 7 surveys the related work, and Section 8 concludes.

Due to space limitations, most proofs of the results in this paper are either omitted or only sketched. Full proofs are available in the supplementary material for referees.

2. Motivating examples

Here, we show two examples in order to illustrate how the biabduction problem arises in the context of verifying array programs, using ASL as the underlying assertion language. In these examples, we often use a ternary *base-offset* variant of the basic array predicate: $\text{array}(a, i, j)$ is syntactic sugar for $\text{array}(a + i, a + j)$. (See Remark 3.4 for further details.)

We note that, in separation logic, Hoare triples $\{A\} C \{B\}$ have a *fault-avoiding* interpretation, where the precondition A guarantees that the code C is memory safe [39].

Example 2.1. A message m of size k must be inserted at the beginning of a buffer b of size n , shifting the previous contents to the right, via the following C function `shift_put`.

```
void shift_put(char *m, int k, char *b, int n){
    memmove(b+k, b, n-k);      // c1
    memcpy(b, m, k);           // c2
}
```

The procedure `memmove(d, s, z)` copies a byte sequence of length z starting from address s into the region starting at address d , even when the two regions overlap. The call c_1 should shift the previous contents of length $n - k$ from the beginning of the buffer to its end. The relevant specification of the call c_1 is the Hoare triple $\{A\} c_1 \{A\}$ where (assuming non-negative ints) A is the following assertion in ASL:

$$k < n : \text{array}(b, 0, k - 1) * \text{array}(b, k, n - 1) .$$

Similarly, `memcpy(d, s, z)` copies z bytes from s into d but overlap is forbidden. The call c_2 above is specified by the triple $\{B\} c_2 \{B\}$, where B is the ASL formula

$$\text{array}(m, 0, k - 1) * \text{array}(b, 0, k - 1) .$$

It is easy to see that $A \not\models B$, so we cannot immediately combine these specifications to prove `shift_put`. To overcome this, we solve the *biabduction* problem: find formulas X, Y such that $A * X \models B * Y$. One possible solution is

$$X = \text{array}(m, 0, k - 1) , \quad Y = \text{array}(b, k, n - 1) .$$

Using $A * X$ as the precondition of $c_1; c_2$ and the knowledge that $A * X \models B * Y$ allows us to apply the derivation given earlier and *automatically abduce* the valid specification

$$\{D\} \text{shift_put}(m, k, b, n) \{D\}$$

where (after merging arrays from b) D is the ASL assertion

$$k < n : \text{array}(m, 0, k - 1) * \text{array}(b, 0, n - 1) .$$

Example 2.2. Here we show how to assemble a valid memory specification for BUILD-MAX-HEAP, an essential preparatory step in *heapsort* (see e.g. [15]), starting from “small specs” for its atomic commands.

```
void BUILD-MAX-HEAP(int n){
  int i = 1; T[i] = b1;      // root
  while (2*i <= n) {
    T[2*i] = b2i;           // left-hand child
    H(2*i);                  // restore max-heap
    if (2*i+1 <= n) {
      T[2*i+1] = b2i+1;     // right-hand child
      H(2*i+1);              // restore max-heap
    }
    i = i+1;
  }
}
```

Given a list of values b_1, \dots, b_n , we insert them one-by-one into the array T , which is viewed as a *binary tree*: for any index i , the indices $2i$ and $2i + 1$ are its left and right ‘children’. Additionally, the values stored in the array should satisfy the *max-heap property*: for every child j with parent i say (so $i = \lfloor j/2 \rfloor$), we have $T[j] \leq T[i]$. This property is maintained using the auxiliary function $H(k)$, which swaps up the newly added $T[k]$ to the proper place along the path from k to the root, 1:

```
void H(int k){
  int j2 = k; int j1 = j2/2;
  while ( j1 >= 1 && T[j1] < T[j2] ){
    swap(T[j1], T[j2]); j2 = j1; j1 = j2/2; }
}
```

Fixing a base-offset a for the array T , observe first that any command of the form $T[j] = b_j$; obeys the obvious memory spec:

$$\{\text{array}(a, j, j)\} T[j] = b_j; \{a + j \mapsto b_j\}$$

Now, writing C_j for a command of the form $T[j] = b_j$;, observe that we can compose a specification for $C_1; C_2$;, as in the previous example, by solving the following biabduction problem: find X and Y such that

$$a + 1 \mapsto b_1 * X \models \text{array}(a, 2, 2) * Y .$$

A minimal solution is quite evident: take $X = \text{array}(a, 2, 2)$ and $Y = \text{array}(a, 1, 1)$. Following the method outlined in the introduction, we can (automatically) generate the valid specification:

$$\{\text{array}(a, 1, 1) * X\} C_1; C_2; \{a + 2 \mapsto b_2 * Y\}$$

Taking into account that $a + 2 \mapsto b_2 \models \text{array}(a, 2, 2)$, and joining arrays, we get

$$\{\text{array}(a, 1, 2)\} C_1; C_2; \{\text{array}(a, 1, 2)\}$$

As $H(2)$ manipulates $T[1]$ and $T[2]$ only, we can show that

$$\{\text{array}(a, 1, 2)\} C_1; C_2; H(2); \{\text{array}(a, 1, 2)\}$$

By iterating this process, we get a valid specification for the unfolded BUILD-MAX-HEAP(n):

$$\begin{aligned} & \{\text{array}(a, 1, n)\} \\ & C_1; C_2; H(2); C_3; H(3); \dots C_n; H(n); \\ & \{\text{array}(a, 1, n)\} \end{aligned}$$

which is a valid specification for BUILD-MAX-HEAP(n):

$$\{\text{array}(a, 1, n)\} \text{BUILD-MAX-HEAP}(n) \{\text{array}(a, 1, n)\}$$

3. Array separation logic, ASL

In this section we present separation logic for arrays, ASL, which employs a similar *symbolic heap* formula structure to that in [4], but which treats contiguous *arrays* in memory rather than linked list segments; we additionally allow a limited amount of pointer arithmetic, given by a conjunction of atomic Presburger formulas.

Definition 3.1 (Symbolic heap). *Terms t , pure formulas Π and spatial formulas F are given by the following grammar:*

$$\begin{aligned} t &::= x \mid n \mid t + t \mid nt \\ \Pi &::= t = t \mid t \neq t \mid t \leq t \mid t < t \mid \Pi \wedge \Pi \\ F &::= \text{emp} \mid t \mapsto t \mid \text{array}(t, t) \mid F * F \\ SH &::= \exists z. \Pi : F \end{aligned}$$

where x ranges over an infinite set Var of *variables*, z over sets of variables, and n over natural number constants in \mathbb{N} . A *symbolic heap* is given by $\exists z. \Pi : F$, where z is a tuple of (distinct) variables, F is a spatial formula and Π is a pure formula. Whenever one of Π, F is empty, we omit the colon. We write $FV(A)$ for the set of free variables occurring in a symbolic heap A .

If $A = \exists z. \Pi : F$ is a symbolic heap, then we write $\text{qf}(A)$ for $\Pi : F$, the *quantifier-free* part of A .

We interpret the above language in a simple stack-and-heap model, in which we take both locations and values to be natural numbers. A *stack* is a function $s : \text{Var} \rightarrow \mathbb{N}$. We extend stacks to interpret terms in the obvious way:

$$s(n) = n, \quad s(t_1 + t_2) = s(t_1) + s(t_2), \quad \text{and} \quad s(nt) = ns(t) .$$

If s is a stack, $z \in \text{Var}$ and $m \in \mathbb{N}$, we write $s[z \mapsto v]$ for the stack defined as s except that $s[z \mapsto v](z) = v$. We extend stacks pointwise to act on tuples of terms.

A *heap* is a finite partial function $h : \mathbb{N} \rightarrow_{\text{fin}} \mathbb{N}$ mapping finitely many locations to values; we write $\text{dom}(h)$ for the

$s, h \models t_1 \sim t_2$	$\Leftrightarrow s(t_1) \sim s(t_2) \quad (\sim \in \{=, \neq, <, \leq\})$
$s, h \models \Pi_1 \wedge \Pi_2$	$\Leftrightarrow s, h \models \Pi_1 \text{ and } s, h \models \Pi_2$
$s, h \models \text{emp}$	$\Leftrightarrow h = e$
$s, h \models t_1 \mapsto t_2$	$\Leftrightarrow \text{dom}(h) = \{s(t_1)\} \text{ and } h(s(t_1)) = s(t_2)$
$s, h \models \text{array}(t_1, t_2)$	$\Leftrightarrow s(t_1) \leq s(t_2) \text{ and } \text{dom}(h) = \{s(t_1), \dots, s(t_2)\}$
$s, h \models F_1 * F_2$	$\Leftrightarrow \exists h_1, h_2. h = h_1 \circ h_2 \text{ and } s, h_1 \models F_1 \text{ and } s, h_2 \models F_2$
$s, h \models \exists z. \Pi : F$	$\Leftrightarrow \exists \mathbf{m} \in \text{Val}^{ z }. s[z \mapsto \mathbf{m}], h \models \Pi \text{ and } s[z \mapsto \mathbf{m}], h \models F$

Figure 1. The ASL satisfaction relation (cf. Defn 3.2).

set of locations on which h is defined, and e for the empty heap that is undefined on all locations. We write \circ for *composition* of domain-disjoint heaps: if h_1 and h_2 are heaps, then $h_1 \circ h_2$ is the union of h_1 and h_2 when $\text{dom}(h_1)$ and $\text{dom}(h_2)$ are disjoint, and undefined otherwise.

Definition 3.2. The *satisfaction relation* $s, h \models A$, where s is a stack, h a heap and A a symbolic heap, is defined by structural induction on A in Fig. 1.

Satisfaction of pure formulas Π does not depend on the heap; we write $s \models \Pi$ to mean that $s, h \models \Pi$ (for any heap h). We write $A \models B$ to mean that A *entails* B , i.e. that $s, h \models A$ implies $s, h \models B$ for all stacks s and heaps h .

Lemma 3.3. *For all quantifier-free symbolic heaps A , if $s, h \models A$ and $s, h' \models A$, then $\text{dom}(h) = \text{dom}(h')$.*

Remark 3.4. Our array predicate employs *absolute* addressing: $\text{array}(k, \ell)$ denotes an array from k to ℓ . In practice, one often reasons about arrays using *base-offset* addressing, where $\text{array}(b, i, j)$ denotes an array from $b + i$ to $b + j$. We can define such a ternary version of our array predicate, overloading notation, by:

$$\text{array}(b, i, j) \stackrel{\text{def}}{=} \text{array}(b + i, b + j)$$

Conversely, any $\text{array}(k, \ell)$ can be represented in base-offset style as $\text{array}(0, k, \ell)$. The moral is that we may freely switch between absolute and base-offset addressing as desired.

In order to obtain sharper complexity results, we will sometimes confine our attention to symbolic heaps in the following special *two-variable form*.

Definition 3.5. A symbolic heap $\exists z. \Pi : F$ is said to be in *two-variable form* if

- (a) its pure part Π is a conjunction of ‘*difference constraints*’ of the form $x = k$, $x = y + k$, $x \leq y + k$, $x \geq y + k$, $x < y + k$, and $x > y + k$, where x and y are variables, and $k \in \mathbb{N}$; (notice that $x \neq y$ is not here);

- (b) its spatial part F contains only formulas of the form $k \mapsto v$, $\text{array}(a, 0, j)$, $\text{array}(a, 1, j)$, and $\text{array}(k, j, j)$, where v, a , and j are variables, and $k \in \mathbb{N}$.

Remark 3.6. The unrestricted pure part of our language is already NP-hard. However, when we restrict pure formulas to conjunctions of ‘difference constraints’¹ as in Definition 3.5, their satisfiability can be decided in polynomial time [15]. Therefore, restricting our symbolic heaps to two-variable form readdresses the challenge of establishing relevant lower bounds to the spatial part of the language.

4. Satisfiability in ASL

Here, we show that *satisfiability* in ASL is NP-complete. This stands in contrast to the situation for symbolic-heaps over list segments, where satisfiability is polynomial [14], and over general inductive predicates, where it is EXP-complete [9]. The problem is stated formally as follows:

Satisfiability problem for ASL. *Given symbolic heap A , decide whether there is a stack s and heap h with $s, h \models A$. (W.l.o.g., we may consider A to be quantifier-free.)*

First, we show that satisfiability of a symbolic heap can be encoded as a Σ_1^0 formula of *Presburger arithmetic*.

Definition 4.1. *Presburger arithmetic* (PbA) is defined as the first-order theory of the natural numbers \mathbb{N} over the signature $\langle 0, s, + \rangle$, where s is the successor function, and 0 and $+$ have their usual interpretations. It is immediate that the relations \neq , \leq and $<$ can be encoded (possibly introducing an existential quantifier), as can the operation of multiplication by a constant.

Note that a stack is just a standard first-order valuation, and that any pure formula in ASL is also a formula of PbA. Moreover, the satisfaction relations for ASL and PbA coincide on such formulas. Thus, we overload \models to include the standard first-order satisfaction relation of PbA.

The intuition behind our encoding of ASL satisfiability in PbA is simple: a symbolic heap is satisfiable exactly when the pure part is satisfiable, each array is well-defined, and all pointers and arrays are non-overlapping with all of the others. For simplicity of exposition, we do this by abstracting away pointers with single-cell arrays.

Definition 4.2. Let A be a quantifier-free symbolic heap, written (without loss of generality) in the form:

$$\Pi : \bigstar_{i=1}^n \text{array}(a_i, b_i) * \bigstar_{i=1}^m c_i \mapsto d_i.$$

We define its *array abstraction* $\lfloor A \rfloor$ as

$$\Pi : \bigstar_{i=1}^n \text{array}(a_i, b_i) * \bigstar_{i=1}^m \text{array}(c_i, c_i).$$

Lemma 4.3. *Let A be a quantifier-free symbolic heap and s a stack. Then $\exists h. s, h \models A \Leftrightarrow \exists h'. s, h' \models \lfloor A \rfloor$.*

¹ The first order theory of such constraints is sometimes called “difference logic” or, amusingly enough, “separation logic” [37]!

Definition 4.4. Let A be a quantifier-free symbolic heap, and let $\lfloor A \rfloor$ be of the form $\Pi : \bigstar_{i=1}^n \text{array}(a_i, b_i)$. We define a corresponding formula $\gamma(A)$ of PbA as

$$\gamma(A) =_{\text{def}} \Pi \wedge \bigwedge_{1 \leq i \leq n} a_i \leq b_i \wedge \bigwedge_{1 \leq i < j \leq n} (b_i < a_j) \vee (b_j < a_i).$$

Note that $\gamma(A)$ is defined in terms of the abstraction $\lfloor A \rfloor$.

Lemma 4.5. For any stack s and any quantifier-free symbolic heap A , we have $s \models \gamma(A) \Leftrightarrow \exists h. s, h \models A$.

Proposition 4.6. Satisfiability for ASL is in NP.

Proof. Letting \mathbf{x} be a tuple of all free variables of a symbolic heap A , the Σ_1^0 Presburger arithmetic sentence $\exists \mathbf{x}. \gamma(A)$, where γ is given by Definition 4.4, is of size quadratic in the size of A . By Lemma 4.5, A is satisfiable iff $\exists \mathbf{x}. \gamma(A)$ is satisfiable. Since the satisfiability problem for Σ_1^0 Presburger arithmetic is in NP [34], so is satisfiability for ASL. \square

Remark 4.7. Symbolic-heap separation logic on list segments [4] enjoys the *small model property*: any satisfiable formula A has a model of size polynomial in the size of A [3]. Unfortunately, this property fails for ASL. E.g., let A_n be a symbolic heap of the form

$$(d_0 = 1) \wedge \bigwedge_{i=0}^{n-1} (d_i < d_{i+1}) : \bigstar_{i=0}^n \text{array}(d_i, 0, d_i).$$

Then we have that $\bigwedge_{i=0}^{n-1} (s(d_{i+1}) > 2s(d_i))$ for any model (s, h) of A_n , which implies that $s(d_n) > 2^n$, and so h occupies a contiguous memory block of at least 2^n cells.

We establish that satisfiability is in fact NP-hard by reduction from the 3-partition problem [18].

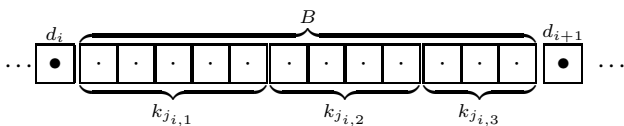
3-partition problem [18]. Given a bound $B \in \mathbb{N}$ and a sequence of natural numbers $\mathcal{S} = (k_1, k_2, \dots, k_{3m})$ such that $\sum_{j=1}^{3m} k_j = mB$, and, in addition, $B/4 < k_j < B/2$ for all $1 \leq j \leq 3m$, decide whether there is a partition of the elements of \mathcal{S} into m groups of three, say

$$\{(k_{j_{i,1}}, k_{j_{i,2}}, k_{j_{i,3}}) \mid 1 \leq i \leq m\},$$

such that $k_{j_{i,1}} + k_{j_{i,2}} + k_{j_{i,3}} = B$ for all $1 \leq i \leq m$.

Definition 4.8. Given an instance (B, \mathcal{S}) of the 3-partition problem, we define a corresponding symbolic heap $A_{B,\mathcal{S}}$.

For convenience, we use the ternary “base-offset” version of our arrays to define $A_{B,\mathcal{S}}$, as given by Remark 3.4. First we introduce $(m+1)$ variables d_i and $3m$ variables a_j . The idea is that the d_i act as single-cell delimiters between chunks of memory of length B , while the a_j serve to allocate arrays of length k_j in the space between some pair of delimiters d_i and d_{i+1} . The arrangement is as follows:



Concretely, $A_{B,\mathcal{S}}$ is the following symbolic heap:

$$\begin{aligned} & \bigwedge_{i=1}^m (d_{i+1} = d_i + B + 1) \wedge \\ & \bigwedge_{j=1}^{3m} (d_1 \leq a_j) \wedge (a_j + k_j < d_{m+1}) : \\ & \bigstar_{i=1}^{m+1} \text{array}(d_i, 0, 0) * \bigstar_{j=1}^{3m} \text{array}(a_j, 1, k_j). \end{aligned}$$

where the indexed “big star” notation abbreviates a sequence of $*$ -conjoined formulas. We observe that $A_{B,\mathcal{S}}$ is quantifier-free and in two-variable form (cf. Defn. 3.5).

Lemma 4.9. Given a 3-partition problem instance (B, \mathcal{S}) , and letting $A_{B,\mathcal{S}}$ be the symbolic heap given by Defn. 4.8,

$A_{B,\mathcal{S}}$ is satisfiable $\Leftrightarrow \exists$ complete 3-partition of \mathcal{S} (w.r.t. B).

Theorem 4.10. The satisfiability problem for ASL is NP-complete, even for quantifier-free and \mapsto -free symbolic heaps in two-variable form.

Proof. Proposition 4.6 provides the upper bound. For the lower bound, Defn. 4.8 and Lemma 4.9 establish a polynomial reduction from the 3-partition problem. \square

5. Biabduction

In this section, we turn to the central focus of this paper, *biabduction* for ASL. In stating this problem, it is convenient to first lift the connective $*$ to symbolic heaps, as follows:

$$(\exists \mathbf{x}. \Pi : F) * (\exists \mathbf{y}. \Pi' : F') = \exists \mathbf{x} \cup \mathbf{y}. \Pi \wedge \Pi' : F * F',$$

where we assume that the existentially quantified variables \mathbf{x} and \mathbf{y} are disjoint, and that no free variable capture occurs (this can always be avoided by α -renaming).

Biabduction problem for ASL. Given satisfiable symbolic heaps A and B , find symbolic heaps X and Y such that $A * X$ is satisfiable and $A * X \models B * Y$.

We first consider quantifier-free biabduction, i.e., where all of A, B, X, Y are quantifier-free (Sec. 5.1). The complexity of quantifier-free biabduction is investigated in Sec. 5.2. We then show that when quantifiers appear in B, Y which are appropriately restricted, existence of solutions can be decided using the machinery we provide for the quantifier-free case (Sec. 5.3). In the same section we also characterise the complexity of biabduction in the presence of quantifiers.

5.1 An algorithm for quantifier-free biabduction

We now present an algorithm for quantifier-free biabduction. Let (A, B) be a biabduction problem and (X, Y) a solution. The intuition is that a model (s, h) of both A and B induces a total order over the terms of A, B , dictating the form of the solution (X, Y) .

Consider Fig. 2, which depicts a biabduction instance (A, B) and a solution (X, Y) (in hatched pattern), where all

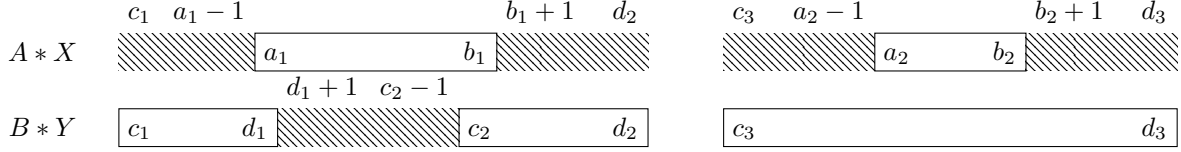


Figure 2. Example showing solutions in Defn. 5.6. Arrays of A, B are displayed as boxes and arrays in X, Y as hatched rectangles.

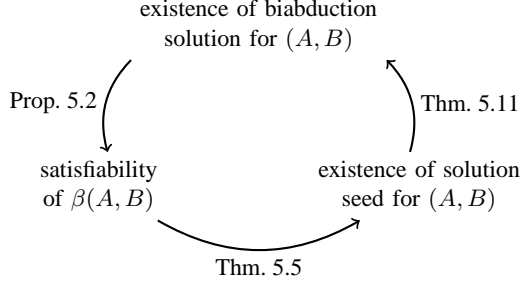


Figure 3. Results on quantifier-free biabduction.

array endpoints in A, B are totally ordered (on the horizontal axis). Using this order, we can compute X, Y , by covering parts that B requires but A does not provide (X) and by covering parts that A requires but B does not provide (Y).

We capture this intuition by (a) defining a PbA formula $\beta(A, B)$ which is shown to be satisfiable whenever there is a solution for the biabduction problem (A, B) (Defn. 5.1, Prop. 5.2); (b) showing that if $\beta(A, B)$ is satisfiable then there exists a formula Δ capturing the total order over the terms of A, B , which we call a *solution seed* (Defn. 5.3, Thm. 5.5); and (c) showing that if there is a solution seed Δ then we can generate a solution X, Y for the biabduction problem (A, B) (Defn. 5.6, Thm. 5.11). These results and the way they compose are shown in Figure 3.

Finally, we show that the problem of finding a solution to a biabduction problem is in NP and that our algorithm is complexity-optimal (Prop. 5.14).

Definition 5.1 (The formula β). Let (A, B) be an instance of the biabduction problem, where

$$A = \Pi : \bigstar_{i=1}^n \text{array}(a_i, b_i) * \bigstar_{i=1}^k t_i \mapsto u_i$$

$$B = \Pi' : \bigstar_{i=1}^m \text{array}(c_i, d_i) * \bigstar_{i=1}^\ell v_i \mapsto w_i$$

We define a formula $\beta(A, B)$ of PbA as follows:

$$\beta(A, B) =_{\text{def}} \gamma(A) \wedge \gamma(B) \wedge \bigwedge_{j=1}^\ell \bigwedge_{i=1}^n (v_j < a_i \vee v_j > b_i) \wedge \bigwedge_{i=1}^\ell \bigwedge_{j=1}^k (t_i \neq v_j \vee u_i = w_j)$$

Proposition 5.2. *If the biabduction problem (A, B) has a solution, then $\beta(A, B)$ is satisfiable.*

Proof. (Sketch) If X, Y is a solution for the problem (A, B) , then any model s, h of $A * X$ (which exists, by assumption) is also a model of $B * Y$. We then show that $s \models \beta(A, B)$, using Lemma 4.5 for the first conjunct of β , and the assumption that $A * X \models B * Y$ for the second and third conjuncts. \square

Given a biabduction problem of the form in Defn. 5.1, we define a set of terms, $\mathcal{T}_{A,B}$, by:

$$\mathcal{T}_{A,B} =_{\text{def}} T(A) \cup T(B) \cup \{b_i + 1 \mid i \in [1, n]\} \cup \{d_i + 1 \mid i \in [1, m]\} \cup \{t_i + 1 \mid i \in [1, k]\} \cup \{v_i + 1 \mid i \in [1, \ell]\}$$

where $T(-)$ denotes the set of all terms in a symbolic heap.

Definition 5.3 (Solution seed). A *solution seed* for a biabduction problem (A, B) in the form of Defn. 5.1 is a pure formula $\Delta = \bigwedge_{i \in I} \delta_i$ such that:

1. Δ is satisfiable, and $\Delta \models \beta(A, B)$;
2. for all $i \in I$, the conjunct δ_i is of the form $(t < u)$ or $(t = u)$, where $t, u \in \mathcal{T}_{A,B}$;
3. for all $t, u \in \mathcal{T}_{A,B}$, there exists $i \in I$ such that δ_i is $(t < u)$ or $(u < t)$ or $(t = u)$.

Lemma 5.4. *Let Δ be a solution seed for a biabduction problem (A, B) . Δ induces a total order on $\mathcal{T}_{A,B}$: for any $e, f \in \mathcal{T}_{A,B}$, $\Delta \models e < f$ or $\Delta \models e = f$ or $\Delta \models f < e$.*

This lemma justifies abbreviating $\Delta \models e < f$ by $e <_\Delta f$; $\Delta \models e \leq f$ by $e \leq_\Delta f$; and, $\Delta \models e = f$ by $e =_\Delta f$.

Theorem 5.5. *If $\beta(A, B)$ is satisfiable, then there exists a solution seed Δ for the biabduction problem (A, B) .*

Proof. (Sketch) Supposing $s \models \beta(A, B)$, we define Δ as:

$$\Delta =_{\text{def}} \bigwedge_{\substack{e, f \in \mathcal{T}_{A,B} \\ s(e) < s(f)}} e < f \wedge \bigwedge_{\substack{e, f \in \mathcal{T}_{A,B} \\ s(e) = s(f)}} e = f.$$

We then show that Δ satisfies Defn. 5.3. \square

We now present a way to compute a solution (X, Y) given a solution seed Δ . The key ingredient is the *arccov* algorithm, given in Fig. 4. Intuitively, *arccov* takes a solution seed Δ and the endpoints of an array (c_j, d_j) in B , and constructs arrays for X in such a way so that every model of $A * X$ includes a submodel that satisfies $\text{array}(c_j, d_j)$. To do this, arrays in A contribute to the coverage of $\text{array}(c_j, d_j)$

and, in addition, the newly created arrays do not overlap with those of A (or themselves) for reasons of consistency.

Note that in `arrcov` we sometimes need to generate terms denoting the predecessor of the start of an array, even though there is no predecessor function in PbA. We achieve this by introducing primed terms a'_i , and add pure constraints that induce this meaning ($a_i + 1 = a'_i$). This is done on demand by `arrcov` in order to avoid the risk of trying to decrement a zero-valued term, thus obtaining an inconsistent formula.

Definition 5.6 (The formulas X, Y). Let Δ be a solution seed for a biabduction problem (A, B) in the form given in Defn. 5.3. The formulas X, Y are defined as follows:

$$\Theta_X : F_X =_{\text{def}} \bigstar_{j=1}^m \text{arrcov}_{A,\Delta}(c_j, d_j) * \bigstar_{j=1}^\ell \text{ptocov}_{A,\Delta}(v_j, w_j)$$

$$\Theta_Y : F_Y =_{\text{def}} \bigstar_{i=1}^n \text{arrcov}_{B,\Delta}(a_i, b_i) * \bigstar_{i=1}^k \text{ptocov}_{B,\Delta}(t_i, u_i)$$

$$\hat{\Delta} =_{\text{def}} \Delta \wedge \Theta_X \wedge \Theta_Y$$

$$X =_{\text{def}} \hat{\Delta} : F_X \quad Y =_{\text{def}} \hat{\Delta} : F_Y$$

Every quantifier-free formula A of ASL is *precise* [32] (by structural induction): for any model s, h there exists *at most one* subheap h' of h such that $s, h' \models A$. This motivates the following notation. We will write $\llbracket A \rrbracket^{s,h}$ to denote the unique subheap $h' \subseteq h$ such that $s, h' \models A$, when it exists.

Proposition 5.7. Let (A, B) be a biabduction problem of the form shown in Defn. 5.3. Let Δ be a solution seed and terms $e, f \in \mathcal{T}_{A,B}$. The call `arrcovA,Δ(e, f)`:

1. *always terminates, issuing up to $n + k$ recursive calls;*
2. *returns a formula (for some $q \in \mathbb{N}$ and sets $I, J \subseteq \mathbb{N}$)*

$$\bigwedge_{i \in I} a_i = a'_i + 1 \wedge \bigwedge_{i \in J} t_i = t'_i + 1 : \bigstar_{i=1}^q \text{array}(l_i, r_i)$$

where for all $i \in [1, q]$, $l_i \in \mathcal{T}_{A,B}$;

3. *for every $i \in [1, q]$, $\hat{\Delta} \models e \leq l_i \leq r_i \leq f$;*
4. *for every $i \in [1, q - 1]$, $\hat{\Delta} \models r_i < l_{i+1}$.*

Lemma 5.8. Let (A, B) be a biabduction instance, Δ a solution seed and X as in Defn. 5.6. Then, $A * X$ is satisfiable.

Proof. (Sketch) We first obtain a stack s by unpacking Defn. 5.3. We extend it to primed terms a'_i, t'_i, c'_i, v'_i in a way that preserves satisfaction of Δ . Using this stack s , we then define appropriate heaps for the constituent parts of A and X and show that they are pairwise disjoint, thus constructing a heap that satisfies $A * X$. \square

Definition 5.9 (The sequences $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$). Let (A, B) be a biabduction problem, Δ a solution seed, X, Y as defined in 5.6 and s, h a model such that $s, h \models A * X$. Then

we define the following sequences $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$ of subheaps of h , such that:

$$\begin{aligned} \mathcal{B}_i^{\text{arr}} &=_{\text{def}} \llbracket \text{array}(c_i, d_i) \rrbracket^{s,h} & i \in [1, m] \\ \mathcal{B}_i^{\text{pto}} &=_{\text{def}} \llbracket v_i \mapsto w_i \rrbracket^{s,h} & i \in [1, \ell] \\ \mathcal{Y}_i^{\text{arr}} &=_{\text{def}} \llbracket \text{arrcov}_{B,\Delta}(a_i, b_i) \rrbracket^{s,h} & i \in [1, n] \\ \mathcal{Y}_i^{\text{pto}} &=_{\text{def}} \llbracket \text{ptocov}_{B,\Delta}(t_i, u_i) \rrbracket^{s,h} & i \in [1, k] \end{aligned}$$

Lemma 5.10. All heaps in $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$ exist (that is, they are well-defined). Also,

1. *For any sequence of heaps \mathcal{S} of $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$, and any distinct $i, j \in [1, |\mathcal{S}|]$, $\mathcal{S}_i \# \mathcal{S}_j$.*
2. *For any two distinct sequences of heaps \mathcal{S}, \mathcal{T} of $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$, and any i, j , $\mathcal{S}_i \# \mathcal{T}_j$.*
3. $\text{dom}(h) \subseteq \bigcup_{i=1}^m \mathcal{B}_i^{\text{arr}} \cup \bigcup_{i=1}^\ell \mathcal{B}_i^{\text{pto}} \cup \bigcup_{i=1}^n \mathcal{Y}_i^{\text{arr}} \cup \bigcup_{i=1}^k \mathcal{Y}_i^{\text{pto}}$.

Theorem 5.11. Given a solution seed Δ for the biabduction problem (A, B) , the formulas X and Y , as computed by Defn. 5.6, form a solution for that instance.

Proof. That (X, Y) is a solution means that $A * X$ is satisfiable and that $A * X \models B * Y$. The first requirement is fulfilled by Lemma 5.8. Here, we show the second.

Let s, h be a model of $A * X$. We need to show that $s, h \models B * Y$. Using Defn. 5.6, we have:

$$A * X = \Pi \wedge \hat{\Delta} : F_{A*X} \quad \text{and} \quad B * Y = \Pi' \wedge \hat{\Delta} : F_{B*Y}$$

It is easy to see that $s \models \Pi' \wedge \hat{\Delta}$: by assumption, $s \models \hat{\Delta}$, and as $\hat{\Delta} \models \Delta$ (Defn. 5.6) and $\Delta \models \gamma(B)$ (Defn. 5.3), it follows that $s \models \Pi'$ as well (Defn. 4.4).

It remains to show that $s, h \models F_{B*Y}$. Recall that $F_{B*Y} = F_B * F_Y$ and that

$$\begin{aligned} F_B &= \bigstar_{i=1}^m \text{array}(c_i, d_i) * \bigstar_{i=1}^\ell v_i \mapsto w_i \\ F_Y &= \bigstar_{i=1}^n \text{arrcov}_{B,\Delta}(a_i, b_i) * \bigstar_{i=1}^k \text{ptocov}_{B,\Delta}(t_i, u_i) \end{aligned}$$

We will do this by (a) defining a subheap $h' \subseteq h$ for each atomic formula σ in F_{B*Y} , such that $s, h' \models \sigma$. Having done this we will need (b) to show that all such subheaps are disjoint, and that (c) their disjoint union equals h .

The sequences $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$ from Defn. 5.9, by construction, fulfil requirement (a) above, given they are well-defined as guaranteed by Lemma 5.10 (main statement). Requirement (b) is covered by items 1 and 2 of Lemma 5.10. Finally, requirement (c) is covered by item 3 of Lemma 5.10. \square

Remark 5.12. The solutions obtained via Defn. 5.6 are constructed from terms in $\mathcal{T}_{A,B}$. This is syntactically optimal in the sense that X, Y are as ‘symbolic’ as A, B are.

Our solutions are potentially stronger than required. Applying Defn. 5.6 to Example 2.1 gives us several solutions,

```

1 Function arrcovA,Δ(e, f)
  Data: a quantifier-free symbolic heap A;
    solution seed Δ; terms e, f in  $\mathcal{T}_{A,B}$ 
  Result: quantifier-free symbolic heap
  // work with  $\mapsto$ -abstraction of A
2 let  $(\Pi : \bigstar_{i=1}^{n+k} \text{array}(\hat{a}_i, \hat{b}_i)) = [A];$ 
3 if  $f <_{\Delta} e$  then
  // nothing to cover
4   return emp;
5 end
6 if  $\exists i \in [1, n+k]. \hat{a}_i \leq_{\Delta} e \leq_{\Delta} \hat{b}_i$  then
  // left endpoint e covered by array( $\hat{a}_i, \hat{b}_i$ )
7   return arrcovA,Δ( $\hat{b}_i + 1, f$ );
8 end
  // left endpoint f not covered
9  $E := \{\hat{a}_j \mid e <_{\Delta} \hat{a}_j \leq_{\Delta} f \text{ for } j \in [1, n+k]\};$ 
10 if  $E = \emptyset$  then
  // no part of array(e, f) covered
11   return array(e, f);
12 end
  // middle of array(e, f) covered by array( $\hat{a}_i, \hat{b}_i$ )
13  $\hat{a}_i := \min_{\Delta}(E);$ 
14 return
  ( $\hat{a}'_i + 1 = \hat{a}_i : \text{array}(e, \hat{a}'_i) * \text{arrcov}_{A,\Delta}(\hat{b}_i + 1, f)$ );

```

```

1 Function ptocovA,Δ(e, f)
2 let  $(\Pi : \bigstar_{i=1}^n \text{array}(a_i, b_i) * \bigstar_{i=1}^k t_i \mapsto u_i) = A;$ 
3 if  $\exists i \in [1, k]. t_i =_{\Delta} e$  then
4   return emp;
5 end
6 if  $\exists i \in [1, n]. a_i \leq_{\Delta} e \leq_{\Delta} b_i$  then
7   return emp;
8 end
9 return  $e \mapsto f;$ 

```

- **Arrays of A / B** appear as boxes with indicated bounds.
- **Arrays of X** appear in a hatched pattern.
- **Recursive calls** appear as dashed boxes with parameters.
- **Terms a'_i** are shown as $a_i - 1$ for readability.

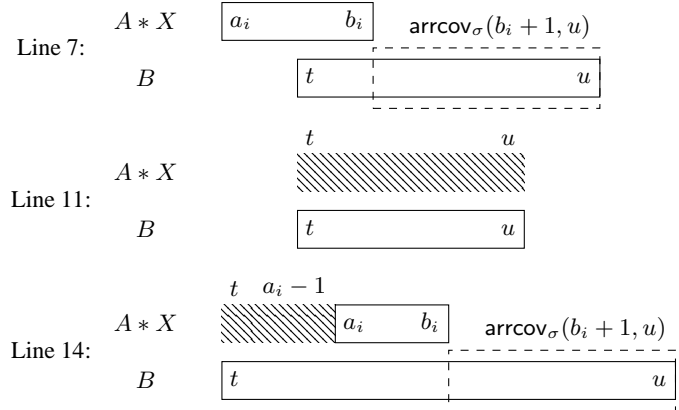


Figure 4. Left: the function $\text{arrcov}_{A,\Delta}(e, f)$. Top right: the function $\text{ptocov}_{A,\Delta}(e, f)$. Bottom right: arrays of A, B, X relevant to each **return** statement in the arrcov function.

corresponding to the number of ways $\text{array}(b, 0, n - 1)$ and $\text{array}(m, 0, k - 1)$ can be situated in memory in relation to each other. However, it can be seen that some solutions can be merged into one, weaker solution. For instance,

$$\begin{aligned}
 X_1 &= m + k \leq b : \text{array}(m, 0, k - 1) \\
 X_2 &= b + n \leq m : \text{array}(m, 0, k - 1)
 \end{aligned}$$

can be merged into the more natural $X = \text{array}(m, 0, k - 1)$.

Our method is, also, complete in the following sense. Suppose (X, Y) is a solution that does not impose a total order over $\mathcal{T}_{A,B}$. Then, there exists a solution (X', Y') computable by our method, such that $X' \models X$ and $Y' \models Y$.

5.2 Complexity of quantifier-free biabduction in ASL

Lemma 5.13. *Let (A, B) be a biabduction instance and Δ a formula satisfying Conditions 2 and 3 of Defn. 5.3. Let $\Gamma = \bigwedge \bigvee \pi$ be a formula where π is of the form $t < u$ or $t = u$ and $t, u \in \mathcal{T}_{A,B}$. Then, checking $\Delta \models \Gamma$ is in PTIME.*

Proposition 5.14. *Deciding if there is a solution for a biabduction problem (A, B) , and constructing it if it exists, can be done in NP.*

Proof. (Sketch) We guess a total order over $\mathcal{T}_{A,B}$ and a polynomially-sized assignment of values s ([34, Theorem 6]) to all terms in $\mathcal{T}_{A,B}$. We convert this order to a formula Δ and check if $s \models \Delta$ (thus showing the satisfiability of Δ) and whether $\Delta \models \beta(A, B)$. If all these conditions hold, we use Defn. 5.6 and obtain formulas X, Y . By Prop. 5.7 and Lemma 5.13 this process runs in PTIME. \square

We establish NP-hardness of quantifier-free biabduction by reduction from the 3-partition problem, similarly to satisfiability in Section 4.

Definition 5.15. Similar to Definition 4.8, given an instance (B, S) of the 3-partition problem, we define corresponding symbolic heaps $\tilde{A}_{B,S}$ and $\tilde{B}_{B,S}$, such that both are satisfiable, quantifier-free, \mapsto -free and in two-variable form. First, $\tilde{A}_{B,S}$ is:

$$\bigwedge_{i=1}^m (d_{i+1} = d_i + B + 1) : \bigstar_{i=1}^{m+1} \text{array}(d_i, 0, 0) .$$

A diagram showing a sequence of boxes. From left to right: an ellipsis, a box containing a black dot, a bracket labeled B spanning six boxes, another box containing a black dot, and a final box followed by an ellipsis.

$$\bigwedge_{i=1}^m (d_{i+1} > d_i) \wedge \bigwedge_{j=1}^{3m} (d_1 \leq a_j) \wedge (a_j + k_j < d_{m+1}):$$

$$\bigstar_{i=1}^{m+1} \text{array}(d_i, 0, 0) * \bigstar_{j=1}^{3m} \text{array}(a_j, 1, k_j).$$

Diagram illustrating an unrestricted sequence structure. The sequence consists of a series of boxes. The first box is empty, followed by a box containing a black dot. Then, a bracket labeled "unrestricted" spans a sequence of boxes. The first box in this bracketed sequence is empty, followed by three boxes each containing a dot. This is followed by a bracket labeled $k_{j_{i,1}}$ spanning the first three dotted boxes, then a bracket labeled $k_{j_{i,2}}$ spanning the next three dotted boxes, and then a bracket labeled $k_{j_{i,3}}$ spanning the next three dotted boxes. The sequence ends with a box containing a black dot and an ellipsis.

$$\beta(\tilde{A}_{B,S}, \tilde{B}_{B,S}) \equiv \gamma(A_{B,S}) .$$

Proof. By reduction from the 3-partition problem (see Section 4). Given an instance (B, S) of this problem, let $A_{B,S}$, $\tilde{A}_{B,S}$, and $\tilde{B}_{B,S}$ be the symbolic heaps given by Defns. 4.8 and 5.15. Note that $\tilde{A}_{B,S}, \tilde{B}_{B,S}$ are satisfiable, quantifier- and \mapsto -free, and in two variable form. Then we have

- This completes the reduction. \square

We use $c_{i,1}$ to denote one of the colours, 1, 2, or 3, the vertex v_i is marked by. We mark also each edge (v_i, v_j) by \widetilde{c}_{ij} , the colour “complementary” to $c_{i,1}$ and $c_{j,1}$.

As for the leaves v_i , we introduce k distinct numbers d_1, \dots, d_k so that the value c_i stored in the location d_i can be used subsequently to identify the colour $c_{i,1}$ marking v_i , e.g., with the help of $(c_{i,1} - 1 \equiv c_i \pmod{3})$.

To encode the fact that no two adjacent vertices v_i and v_j share the same colour, we use $c_{i,1}$, $c_{j,1}$, and \widetilde{c}_{ij} as the addresses for three consecutive cells within a memory chunk of length 3 given by $\text{array}(e_{ij}, 1, 3)$, which forces these colours to form a *permutation* of $(1, 2, 3)$. (The base-offset addresses e_{ij} are chosen to ensure that all the arrays in question are disjoint.)

Concretely, we define A_G to be the following symbolic heap:

$$\bigstar_{i=1}^k \text{array}(d_i, 1, 1) * \bigstar_{(v_i, v_j) \in E} \text{array}(e_{ij}, 1, 3).$$

We define B_G as follows:

$$\begin{aligned} \exists \mathbf{z}. & \left(\bigwedge_{i=1}^n (1 \leq c_{i,1} \leq 3) \wedge \bigwedge_{(v_i, v_j) \in E} (1 \leq \widetilde{c}_{ij} \leq 3) \right. \\ & \wedge \bigwedge_{i=1}^k (c_{i,1} - 1 \equiv c_i \pmod{3}) : \\ & \bigstar_{i=1}^k d_i \mapsto c_i * \bigstar_{(v_i, v_j) \in E} \text{array}(e_{ij}, c_{i,1}, c_{i,1}) \\ & * \bigstar_{(v_i, v_j) \in E} \text{array}(e_{ij}, c_{j,1}, c_{j,1}) * \text{array}(e_{ij}, \widetilde{c}_{ij}, \widetilde{c}_{ij}) \big). \end{aligned}$$

where the existentially quantified variables \mathbf{z} are all variables occurring in B_G that are not mentioned explicitly in A_G .

Lemma 5.20. *Let G be a 2-round 3-colouring instance. The biabduction problem (A_G, B_G) has a solution iff there is a winning strategy for colouring G , where A_G and B_G are the symbolic heaps given by Defn. 5.19.*

Theorem 5.21. *The biabduction problem (A, B) for ASL is Π_2^P -hard, even if A is quantifier-free and \mapsto -free.*

Proof. Follows from Lemma 5.20. \square

6. Entailment

In this section, we investigate the *entailment* problem for ASL. We establish an upper bound of Π_1^{EXP} in the *weak EXP hierarchy* [24] via an encoding into Π_2^0 PbA, and a lower bound of Π_2^P in the *polynomial-time hierarchy* [35]. Moreover, for quantifier-free entailments, we show that the problem becomes coNP-complete.

Entailment problem for ASL. *Given symbolic heaps A and B , decide whether $A \models B$.*

As in the biabduction problem, A may be considered quantifier-free, but the existential quantifiers in B may not mention any variable appearing in the RHS of a \mapsto -formula.

The intuition underlying our encoding of entailment into Presburger arithmetic is as follows: There exists a countermodel for $A \models B$ iff there exists a stack s that induces a model for A (captured by $\gamma(A)$ from Defn. 4.4 / Lemma 4.5)

and, for every instantiation of the existentially quantified variables in B (say \mathbf{z}), one of the following holds under s :

1. the quantifier-free body $\text{qf}(B)$ of B becomes unsatisfiable (captured by $\neg\gamma(\text{qf}(B))$); or
2. some heap location is covered by an array or pointer in A , but not by any array or pointer in B , or vice versa; or
3. the LHS of some pointer in B is covered by an array in A (and therefore we can choose the contents of the array different to the “correct” data contents of the pointer); or
4. some pointer in B is covered by a pointer in A , but their data contents disagree.

Similar to Prop. 5.18, this intuition also explains the reason for our restriction on existential quantification in the entailment problem: if we are allowed to quantify over the RHS of \mapsto formulas, then item 3 above might or might not be sufficient to construct a countermodel. For example, there is a countermodel for $\text{array}(x, x) \models \exists y. y \leq 3 : x \mapsto y$, and for $\text{array}(x, x) \models x \mapsto y$, but not for $\text{array}(x, x) \models \exists y. x \mapsto y$.

Definition 6.1. Let A and B be \mapsto -free symbolic heaps, with spatial parts as follows:

$$\begin{aligned} A : & \text{array}(a_1, b_1) * \dots * \text{array}(a_n, b_n) \\ B : & \text{array}(c_1, d_1) * \dots * \text{array}(c_m, d_m) \end{aligned}$$

Then we define the formula $\phi(A, B)$ of PbA to be

$$\exists x. \bigvee_{i=1}^n a_i \leq x \leq b_i \wedge \bigwedge_{j=1}^m (x < c_j) \vee (x > d_j),$$

where x is a fresh variable. We lift $\phi(-, -)$ to arbitrary symbolic heaps by $\phi(A, B) = \phi(\lfloor \text{qf}(A) \rfloor, \lfloor \text{qf}(B) \rfloor)$, i.e. by ignoring quantifiers and abstracting pointers to arrays using $\lfloor - \rfloor$ from Defn. 4.2.

Lemma 6.2. *We can rewrite $\phi(A, B)$ as a quantifier-free formula at only polynomial cost.*

Definition 6.3. Let A and B be symbolic heaps with A quantifier-free:

$$\begin{aligned} A : & \Pi : \bigstar_{i=1}^n \text{array}(a_i, b_i) * \bigstar_{i=1}^k t_i \mapsto u_i \\ B : & \exists \mathbf{z}. \Pi' : \bigstar_{j=1}^m \text{array}(c_j, d_j) * \bigstar_{j=1}^\ell v_j \mapsto w_j \end{aligned}$$

where the existentially quantified variables \mathbf{z} are disjoint from all variables in A . We define formulas $\psi_1(A, B)$, $\psi_2(A, B)$ and $\chi(A, B)$ of PbA as follows:

$$\begin{aligned} \psi_1(A, B) &= \bigvee_{i=1}^n \bigvee_{j=1}^\ell a_i \leq v_j \leq b_i, \\ \psi_2(A, B) &= \bigvee_{i=1}^k \bigvee_{j=1}^\ell (t_i = v_j) \wedge (u_i \neq w_j), \text{ and} \\ \chi(A, B) &= \gamma(A) \wedge \forall \mathbf{z}. (\neg\gamma(\text{qf}(B)) \vee \phi(A, B) \\ &\quad \vee \phi(B, A) \vee \psi_1(A, B) \vee \psi_2(A, B)) \end{aligned}$$

where $\gamma(-)$ is the encoding of satisfiability (Defn. 4.4), and $\phi(-, -)$ is given by Defn. 6.1.

Lemma 6.4. For any instance (A, B) of the ASL entailment problem above, and for any stack s ,

$$s \models \chi(A, B) \Leftrightarrow \exists h. s, h \models A \text{ and } s, h \not\models B.$$

Theorem 6.5. Entailment for ASL is in Π_1^{EXP} . If the number of variables in A, B is fixed then the problem is in Π_2^P , and if B is quantifier-free then the problem is in coNP.

Proof. Let A and B be symbolic heaps with A quantifier-free. Letting \mathbf{x} be a list of all free variables in A and B , we observe that $\exists \mathbf{x}. \chi(A, B)$ is a Σ_3^0 PbA sentence of size polynomial in the size of A and B . By Lemma 6.4, we have that $\exists \mathbf{x}. \chi(A, B)$ is satisfiable if and only if $A \not\models B$. Therefore, $A \models B$ if and only if the Π_3^0 PbA sentence $\forall \mathbf{x}. \neg \chi(A, B)$ is satisfiable.

However, according to Lemma 6.2, we can eliminate the existential quantifier from the subformulas $\phi(A, B)$ and $\phi(B, A)$ inside $\chi(A, B)$, still at only polynomial cost. Writing $\chi'(A, B)$ for the formula so obtained, $\forall \mathbf{x}. \neg \chi'(A, B)$ then becomes a Π_2^0 sentence.

Satisfiability in Π_2^0 Presburger arithmetic is in Π_1^{EXP} [23]. If the set of variables in A and B has fixed size k , then the decision sentence above has exactly $k + 1$ quantifiers, in which case satisfiability is in Π_2^P [20]. Finally, if B is quantifier-free, the decision sentence is a Π_1^0 formula and so can be decided in coNP time [35]. \square

In order to obtain the Π_2^P lower bound for entailment, we exhibit a reduction from the same colourability problem as in Section 5.3.

Definition 6.6. (cf. Definition 5.19) Let $G = (V, E)$ be an undirected graph with n vertices and k leaves. To simulate the colourability game on G , we define a pair of \mapsto -free symbolic heaps: a quantifier-free A_G , to encode an arbitrary 3-colouring of the leaves, and an existentially quantified B_G , to encode a perfect 3-colouring of the whole G .

We use $c_{i,1}$ to denote the colour the vertex v_i is marked by. We mark also each edge (v_i, v_j) by \widetilde{c}_{ij} , “complementary” to $c_{i,1}$ and $c_{j,1}$.

We encode the fact that no two adjacent vertices v_i and v_j share the same colour in accordance with Definition 5.19. (The numbers e_{ij} are chosen to ensure that all the arrays in question are disjoint.)

Concretely, we define A_G to be the following symbolic heap:

$$\bigwedge_{i=1}^k (1 \leq c_{i,1} \leq 3) : *_{(v_i, v_j) \in E} \text{array}(e_{ij}, 1, 3).$$

We define B_G as follows:

$$\begin{aligned} \exists \mathbf{z}. \big(\bigwedge_{i=1}^n (1 \leq c_{i,1} \leq 3) \wedge \bigwedge_{(v_i, v_j) \in E} (1 \leq \widetilde{c}_{ij} \leq 3) : \\ *_{(v_i, v_j) \in E} \text{array}(e_{ij}, c_{i,1}, c_{j,1}) \\ * *_{(v_i, v_j) \in E} \text{array}(e_{ij}, c_{j,1}, c_{i,1}) * \text{array}(e_{ij}, \widetilde{c}_{ij}, \widetilde{c}_{ij}) \big). \end{aligned}$$

where the existentially quantified variables \mathbf{z} are all variables occurring in B_G that are not mentioned explicitly in A_G .

Lemma 6.7. Let G be a 2-round 3-colouring instance, and let A_G and B_G be the symbolic heaps given by Defn. 6.6. Then, we have

$$A_G \models B_G \Leftrightarrow \exists \text{ winning strategy for colouring } G.$$

Theorem 6.8. The entailment problem $A \models B$ is Π_2^P -hard, even when all variables are bounded by 3, A is quantifier-free, and A, B are \mapsto -free symbolic heaps in two-variable form. Moreover, the entailment problem is coNP-hard even for quantifier-free symbolic heaps in two-variable form.

Proof. For the general case, Definition 6.6 and Lemma 6.7 establish a reduction from the 2-round 3-colourability problem, which is Π_2^P -hard [1].

For the quantifier-free case, the upper bound is immediate by Thm. 6.5. For the lower bound, consider the entailment

$$A_{B,S} \models x < x : \text{emp}$$

where (B, S) is an instance of the 3-partition problem (see Section 4) and $A_{B,S}$ is the symbolic heap in two-variable form constructed in Defn. 4.8. Using Lemma 4.9, this entailment is valid iff there is *no* complete 3-partition on S w.r.t. B , which is a coNP-hard problem. \square

In the general case, there is a complexity gap between our upper and lower bounds for entailment: $\Pi_1^{\text{EXP}} = \text{coNEXP}^{\text{NP}}$ versus $\Pi_2^P = \text{coNP}^{\text{NP}}$, respectively. It seems plausible that the lower bound is at least EXP: however, an encoding of, e.g., Π_2^0 Presburger arithmetic in ASL does not seem straightforward, because our pure formulas are conjunctions rather than arbitrary Boolean combinations of atomic Presburger formulas.

Nevertheless, we can detect the essential difference between the biabduction and entailment problems for ASL (at least in the case where the existential quantifiers in B are restricted as described above). Namely, by Theorem 6.8 entailment is still Π_2^P -hard whereas, by Props. 5.14 and 5.18, the biabduction problem belongs to NP.

7. Related work

The literature most closely related to our work in the present paper divides, broadly speaking, into four main categories.

Separation logic over linked list segments. Perhaps the most popular and extensively studied part of separation logic is the symbolic heap fragment over linked lists, introduced and shown decidable in [4]. This fragment is essentially the one employed in Facebook’s INFER tool [12]. Here, the pure part of symbolic heaps is a conjunction of simple equalities and disequalities between expressions (typically just variables or the constant nil), while the spatial part admits points-to formulas $E \mapsto E'$, denoting a single pointer in the heap, and list segment formulas of the form $\text{ls } E \ E'$, denoting a linked list in the heap from E to E' .

Following the initial decidability result, it was shown in [14] that satisfiability and entailment in this logic are in PTIME. The biabduction problem for this fragment and practical approaches to it were first studied in [11]; in [19] it was shown that the *abduction* problem (where only an “antiframe” X is computed) is in fact NP-complete.

We observe that this fragment and our ASL are largely disjoint: our arrays cannot be defined in terms of lss, or vice versa, while ASL also employs arithmetic formulas rather than simple (dis)equality constraints. This is also reflected in the differences in their respective complexity bounds.

Separation logic with inductive predicates. There has been substantial research interest in (symbolic heap) separation logic over general *inductively defined predicates* [7], as opposed to fixed data structures such as lists (or indeed arrays). Such predicates can be used to describe arbitrary data structures in memory; they might be provided to an analysis by the user, or perhaps inferred automatically (cf. [8, 28]).

When arbitrary inductive definitions over symbolic heaps are permitted, the entailment problem is undecidable [3] while satisfiability and even model checking (i.e., deciding whether a given stack-heap pair satisfies a given formula) become EXP-complete (cf. [9] resp. [10]). More tractable fragments can be obtained by restricting the admissible forms of inductive definitions. A fragment in which all definitions have *bounded treewidth* [25] was shown to have a decidable entailment problem by reduction to bounded-treewidth monadic second-order logic; a variant of this fragment, with different restrictions, was similarly shown decidable in [38]. However, our ASL cannot be encoded even in the unrestricted fragment, owing to the absence of arithmetic.

Very recently, in [21], decidability of satisfiability and entailment was obtained for a fragment of symbolic-heap separation logic with restricted inductive predicates (called “linearly compositional”) and Presburger arithmetic constraints. However, ASL cannot be encoded in this fragment, because pointers and data variables belong to disjoint sorts, effectively disallowing pointer arithmetic. Moreover, we provide an analysis of biabduction, which is the central focus of our paper, but not considered in [21].

Finally, also very recently, a semidecision procedure for satisfiability in symbolic-heap separation logic with inductive definitions and Presburger arithmetic appeared in [29]. ASL can be encoded in their logic, but, as far as we can tell, not into the subfragment for which they show satisfiability decidable. We note that in any case this decidability result comes without any complexity bounds.

Separation logic with iterated separating conjunction. The *iterated separating conjunction* (ISC) [33], a binding operator for expressing various unbounded data structures, was recognised early on as a way of reasoning about arrays. For example, the ISC was employed recently in a framework for reasoning about memory permissions, with the aim of enabling symbolic execution of concurrent array-manipulating

programs [30]. An earlier paper employing a form of ISC and biabduction is [22], where the aim is to design a bottom-up shape analysis for unannotated code.

However, although our array predicate can be expressed using the ISC, we do not know of any existing decision procedures for biabduction, entailment or even satisfiability in such a logic, which may be of higher complexity or become undecidable (there is certainly no investigation of these issues in either [22] or [30]). Our work is aimed at underpinning compositional analyses of unannotated code; in contrast, the analysis promoted in [30] requires fully annotated programs and does not employ, or investigate, biabduction. As for [22], arrays are not considered and arithmetic is disallowed (even though arrays are expressible with its ISC); therefore array-manipulating programs cannot be treated.

Other program analyses on arrays. A significant amount of research effort has previously focused on the verification of array-manipulating programs either via invariant inference and theorem proving, or via abstract interpretation (for instance [2, 6, 16, 17, 27, 36]). These approaches differ from ours technically, but also in intention. First, the emphasis in these investigations is on data constraints and, thus, tends towards proving general safety properties of programs. Here, we intentionally restrict the language so that we can obtain sound and complete algorithms which can be used for establishing memory safety of programs but not for proving arbitrary safety properties. Second, such approaches are typically whole-program analyses that cannot be used in a bottom-up fashion or on partial programs. In contrast, our focus is on biabduction, one of the key ingredients that makes such a compositional approach possible.

8. Conclusions and future work

In this paper, we investigate ASL, a separation logic aimed at compositional memory safety proofs for array-manipulating programs. We focus on *biabduction*, the key to interprocedural specification inference: we give a sound and complete NP algorithm for biabduction that computes solutions by finding a consistent ordering of the array endpoints, and we show that the problem is NP-hard in the quantifier-free case. In addition, we show that the satisfiability problem for ASL is NP-complete, and entailment is decidable, being coNP-complete for quantifier-free formulas, and at least Π_2^P -hard (perhaps much harder) in general. We believe that ours are the first decision procedures for separation logic over arrays; certainly, we believe that we are the first to treat biabduction in this context.

The obvious direction of travel for future work is to build an abductive program analysis à la INFER [12] for array programs, using ASL as the assertion language. The first step is to implement an algorithm for biabduction. A direct implementation of our algorithm in Section 5.1, using an SMT solver to find a solution seed, is the most immediate possibility, but not the only one; one might also try possibly-

incomplete but fast approaches based on theorem proving (cf. [11]). A currently extant problem is in finding biabduction solutions that are as logically weak as possible; our algorithm currently commits to a total ordering of all arrays even if a partial ordering would be sufficient. We believe that, in practice, this could be resolved by refining the notion of a solution seed so that it carries *just* enough information for computing the spatial formulas in X and Y . A more conceptually interesting problem is how we might assess the quality of logically incomparable biabduction solutions (e.g. according to the amount of memory they occupy).

In addition, a program analysis for ASL will rely not just on biabduction but also on suitable *abstraction* heuristics for discovering loop invariants; this seems an interesting and non-trivial problem for the near future.

Finally, readers might wonder about the possibility of combining ASL with other fragments of separation logic, such as the linked list fragment, for expressivity reasons. Certainly, we expect that some programs might manipulate, e.g., both linked lists *and* arrays at the same time (and possibly other dynamic data structures too), and a combined language would then clearly be needed to reason about such programs. However, it is not clear whether such a logic (with, say, arithmetic constraints, arrays and linked lists) would enjoy good computational properties; a potentially problematic issue is that a heap might simultaneously satisfy, e.g., a $*$ -conjunction of single heap cells, an array *and* a linked list, all at the same time. We consider this a very interesting area for future study.

References

- [1] M. Ajtai, R. Fagin, and L. J. Stockmeyer. The closure of monadic NP. *J. Comput. Syst. Sci.*, 60(3):660–716, 2000.
- [2] F. Alberti, S. Ghilardi, and N. Sharygina. Decision procedures for flat array properties. In *Proc. TACAS-20*, pages 15–30. Springer, 2014.
- [3] T. Antonopoulos, N. Gorogiannis, C. Haase, M. Kanovich, and J. Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In *Proc. FoSSaCS-17*, pages 411–425. Springer, 2014.
- [4] J. Berdine, C. Calcagno, and P. O’Hearn. A decidable fragment of separation logic. In *Proc. FSTCS-24*, volume 3328 of *LNCS*, pages 97–109. Springer, 2004.
- [5] J. Berdine, B. Cook, and S. Ishtiaq. SLayer: memory safety for systems-level code. In *Proc. CAV-23*, pages 178–183. Springer, 2011.
- [6] A. Bouajjani, C. Drăgoi, C. Enea, and M. Sighireanu. Accurate invariant checking for programs manipulating lists and arrays with infinite data. In *Proc. ATVA-10*, volume 7561 of *LNCS*, pages 167–182. Springer-Verlag, 2012.
- [7] J. Brotherston. Formalised inductive reasoning in the logic of bunched implications. In *Proc. SAS-14*, volume 4634 of *LNCS*, pages 87–103. Springer-Verlag, 2007.
- [8] J. Brotherston and N. Gorogiannis. Cyclic abduction of inductively defined safety and termination preconditions. In *Proc. SAS-21*, volume 8723 of *LNCS*, pages 68–84. Springer, 2014.
- [9] J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Pérez. A decision procedure for satisfiability in separation logic with inductive predicates. In *Proc. CSL-LICS*, pages 25:1–25:10. ACM, 2014.
- [10] J. Brotherston, N. Gorogiannis, M. Kanovich, and R. Rowe. Model checking for symbolic-heap separation logic with inductive predicates. In *Proc. POPL-43*, pages 84–96. ACM, 2016.
- [11] C. Calcagno, D. Distefano, P. O’Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. *J. ACM*, 58(6), December 2011.
- [12] C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. O’Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez. Moving fast with software verification. In *Proc. NFM-7*, volume 9058 of *LNCS*, pages 3–11. Springer, 2015.
- [13] W.-N. Chin, C. David, H. H. Nguyen, and S. Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Sci. Comp. Prog.*, 77(9):1006–1036, 2012.
- [14] B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *Proc. CONCUR-22*, volume 6901 of *LNCS*, pages 235–249. Springer, 2011.
- [15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.
- [16] P. Cousot, R. Cousot, and F. Logozzo. A parametric segmentation functor for fully automatic and scalable array content analysis. In *Proc. POPL-38*, pages 105–118. ACM, 2011.
- [17] I. Dillig, T. Dillig, and A. Aiken. Fluid updates: Beyond strong vs. weak updates. In *Proc. ESOP-19*, pages 246–266. Springer, 2010.
- [18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0-7167-1044-7.
- [19] N. Gorogiannis, M. Kanovich, and P. W. O’Hearn. The complexity of abduction for separated heap abstractions. In *Proc. SAS-18*, volume 6887 of *LNCS*, pages 25–42. Springer, 2011.
- [20] E. Grädel. Subclasses of Presburger arithmetic and the polynomial-time hierarchy. *Theoretical Computer Science*, 56:289–301, 1988.
- [21] X. Gu, T. Chen, and Z. Wu. A complete decision procedure for linearly compositional separation logic with data constraints. In *Proc. IJCAR*, volume 9706 of *LNAI*, pages 532–549. Springer, 2016.
- [22] B. S. Gulavani, S. Chakraborty, G. Ramalingam, and A. V. Nori. Bottom-up shape analysis. In *Proc. SAS-16*, pages 188–204. Springer, 2009.
- [23] C. Haase. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *Proceedings of CSL-LICS*, pages 47:1–47:10. ACM, 2014.
- [24] J. Hartmanis, N. Immerman, and V. Sewelson. Sparse sets in NP-P: EXPTIME versus NEXPTIME. *Inform. Control*, 65(2): 158 – 181, 1985.

- [25] R. Iosif, A. Rogalewicz, and J. Simacek. The tree width of separation logic with recursive definitions. In *Proc. CADE-24*, volume 7898 of *LNAI*, pages 21–38. Springer, 2013.
- [26] B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. VeriFast: A powerful, sound, predictable, fast verifier for C and Java. In *Proc. NFM-3*, volume 6617 of *LNCS*, pages 41–55. Springer, 2011.
- [27] L. Kovács and A. Voronkov. Finding loop invariants for programs over arrays using a theorem prover. In *Proc. FASE-12*, pages 470–485. Springer, 2009.
- [28] Q. L. Le, C. Gherghina, S. Qin, and W.-N. Chin. Shape analysis via second-order bi-abduction. In *Proc. CAV-26*, volume 8559 of *LNCS*, pages 52–68. Springer, 2014.
- [29] Q. L. Le, J. Sun, and W.-N. Chin. Satisfiability modulo heap-based programs. In *Proc. CAV-28*, 2016.
- [30] P. Müller, M. Schwerhoff, and A. J. Summers. Automatic verification of iterated separating conjunctions using symbolic execution. In *Proc. CAV-28*, to appear, 2016.
- [31] P. W. O’Hearn. Resources, concurrency and local reasoning. *Theoretical Computer Science*, 375(1–3):271–307, 2007.
- [32] P. W. O’Hearn, H. Yang, and J. C. Reynolds. Separation and information hiding. In *Proc. POPL-31*, pages 268–280. ACM, 2004.
- [33] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. LICS-17*, pages 55–74. IEEE, 2002.
- [34] B. Scarpellini. Complexity of subcases of Presburger arithmetic. *Trans. American Mathematical Society*, 284(1):203–218, 1984.
- [35] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.
- [36] T. Ströder, J. Giesl, M. Brockschmidt, F. Frohn, C. Fuhs, J. Hensel, P. Schneider-Kamp, and C. Aschermann. Automatically proving termination and memory safety for programs with pointer arithmetic. *Journal of Automated Reasoning*. To appear.
- [37] M. Talupur, N. Sinha, O. Strichman, and A. Pnueli. Range allocation for separation logic. In *Proc. CAV-16*, volume 3114 of *LNCS*, pages 148–161. Springer, 2004.
- [38] M. Tatsuta and D. Kimura. Separation logic with monadic inductive definitions and implicit existentials. In *Proc. APLAS-13*, volume 9458 of *LNCS*, pages 69–89. Springer, 2015.
- [39] H. Yang and P. O’Hearn. A semantic basis for local reasoning. In *Proc. FOSSACS-5*, pages 402–416. Springer, 2002.

A. Proofs of results in Section 3

Lemma 3.3. *For all quantifier-free symbolic heaps A , if $s, h \models A$ and $s, h' \models A$, then $\text{dom}(h) = \text{dom}(h')$.*

Proof. Writing $A = \Pi : F$, we proceed by structural induction on the spatial part F .

Case $F = \text{emp}$: By definition, $\text{dom}(h) = \text{dom}(h') = \emptyset$.

Case $F = t_1 \mapsto t_2$: By definition, $\text{dom}(h) = \text{dom}(h') = \{s(t_1)\}$.

Case $F = \text{array}(t_1, t_2)$: By definition, $\text{dom}(h) = \text{dom}(h') = \{s(t_1), \dots, s(t_2)\}$.

Case $F = F_1 * F_2$: We have $h = h_1 \circ h_2$ and $h' = h'_1 \circ h'_2$, where $s, h_1 \models F_1$ and $s, h'_1 \models F_1$, and $s, h_2 \models F_2$ and $s, h'_2 \models F_2$. Since $s, h_1 \models F_1$ and $s, h'_1 \models F_1$, we have $\text{dom}(h_1) = \text{dom}(h'_1)$ by induction hypothesis. Similarly, $\text{dom}(h_2) = \text{dom}(h'_2)$. Because \circ is defined as the union of domain-disjoint heaps, it follows that $\text{dom}(h_1 \circ h_2) = \text{dom}(h_2 \circ h'_2)$. That is, $\text{dom}(h) = \text{dom}(h')$ as required. This completes the induction. \square

B. Proofs of results in Section 4

Lemma 4.3. *Let A be a quantifier-free symbolic heap and s a stack. Then,*

$$\exists h. s, h \models A \Leftrightarrow \exists h'. s, h' \models \lfloor A \rfloor.$$

Proof. Let A and $\lfloor A \rfloor$ be as shown in Defn. 4.2.

(\Rightarrow) Immediate by the semantics of $*$ and the observation that $c_j \mapsto d_j \models \text{array}(c_j, c_j)$ for all $j \in [1, m]$.

(\Leftarrow) Let s, h be a model of $\lfloor A \rfloor$. We define a model s, \hat{h} such that $s, \hat{h} \models A$. First, by assumption we have $s \models \Pi$. Also, there exist disjoint heaps $h_1, \dots, h_n, h'_1, \dots, h'_m$ such that $h = h_1 \circ \dots \circ h_n \circ h'_1 \circ \dots \circ h'_m$ and $s, h_i \models \text{array}(a_i, b_i)$ for $i \in [1, n]$, and $s, h'_j \models \text{array}(c_j, c_j)$ for $j \in [1, m]$. We define new heaps h''_1, \dots, h''_m as follows. The heap h''_j is defined by $\text{dom}(h''_j) = \{s(c_j)\}$ and $h''_j(s(c_j)) = s(d_j)$ for all $j \in [1, m]$. We then define a new heap $\hat{h} = h_1 \circ \dots \circ h_n \circ h''_1 \circ \dots \circ h''_m$, which is well defined by the fact that $\text{dom}(h''_j) = \text{dom}(h'_j)$ and the assumption that h is well defined. It is easy to see that $s, h''_j \models c_j \mapsto d_j$ and by the semantics of $*$ we are done. \square

Lemma 4.5. *For any stack s and any quantifier-free symbolic heap A ,*

$$s \models \gamma(A) \Leftrightarrow \exists h. s, h \models A.$$

Proof. First, note that satisfiability of A coincides with the satisfiability of $\lfloor A \rfloor$ by Lemma 4.3. Thus it suffices to consider the case when A is \mapsto -free. We assume then that $A = \Pi : \bigstar_{i=1}^n \text{array}(a_i, b_i)$, and establish each direction of the lemma separately.

(\Leftarrow) Suppose that $s, h \models A$. That is, $s \models \Pi$ and there exist heaps h_1, \dots, h_n such that $h = h_1 \circ \dots \circ h_n$ and $s, h_i \models \text{array}(a_i, b_i)$ for each $i \in [1, n]$. We require to show that $s \models \gamma(A)$.

First, $s \models \Pi$ by assumption. Next, for each $i \in [1, n]$, we have $s(a_i) \leq s(b_i)$ because $s, h_i \models \text{array}(a_i, b_i)$; hence $s \models \bigwedge_{1 \leq i \leq n} a_i \leq b_i$. Finally, letting $1 \leq i < j \leq n$, we have $\text{dom}(h_i) = \{s(a_i), \dots, s(b_i)\}$ and $\text{dom}(h_j) = \{s(a_j), \dots, s(b_j)\}$. Since $\text{dom}(h_i)$ and $\text{dom}(h_j)$ are disjoint by assumption, we must have either $s(b_i) < s(a_j)$ or $s(b_j) < s(a_i)$, therefore $s \models \bigwedge_{1 \leq i < j \leq n} (b_i < a_j) \vee (b_j < a_i)$. Putting everything together, $s \models \gamma(A)$ as required.

(\Rightarrow) Supposing that $s \models \gamma(A)$, of the form above, we require to construct a heap h such that $s, h \models A$. For each $i \in [1, n]$, define a heap h_i by $\text{dom}(h_i) = \{s(a_i), \dots, s(b_i)\}$ (the contents may be chosen arbitrarily). We observe that $\text{dom}(h_i)$ is well defined because $s \models a_i \leq b_i$ by assumption. By construction, we have $s, h_i \models \text{array}(a_i, b_i)$.

Next, we claim that $h_1 \circ \dots \circ h_n$ is defined. Supposing not (for contradiction), then there exist h_i, h_j with $1 \leq i < j \leq n$ such that $\text{dom}(h_i) \cap \text{dom}(h_j) \neq \emptyset$. That is, $\{s(a_i), \dots, s(b_i)\} \cap \{s(a_j), \dots, s(b_j)\} \neq \emptyset$, which means that (without loss of generality) either $s(a_j)$ or $s(b_j)$ falls within the range $\{s(a_i), \dots, s(b_i)\}$; i.e., either $s(a_i) \leq s(a_j) \leq s(b_i)$ or $s(a_i) \leq s(b_j) \leq s(b_i)$ (or both). By assumption, we have $s \models (b_i < a_j) \vee (b_j < a_i)$, meaning that either $s(b_i) < s(a_j)$ or $s(b_j) < s(a_i)$. This gives us four cases to consider, and it is simple to see that each leads to a contradiction: **(1)** if $s(a_i) \leq s(a_j) \leq s(b_i)$ and $s(b_i) < s(a_j)$, we immediately get $s(a_j) < s(a_j)$; **(2)** if $s(a_i) \leq s(a_j) \leq s(b_i)$ and $s(b_j) < s(a_i)$, we get $s(b_j) < s(a_j)$, contradicting $s \models a_j \leq b_j$; **(3)** if $s(a_i) \leq s(b_j) \leq s(b_i)$ and $s(b_i) < s(a_j)$, we again get $s(b_j) < s(a_j)$; **(4)** if $s(a_i) \leq s(b_j) \leq s(b_i)$ and $s(b_j) < s(a_i)$, we get $s(a_i) < s(a_i)$. Putting everything together, and using the fact that $s \models \Pi$, we obtain $s, h_1 \circ \dots \circ h_n \models A$, and are done. \square

Lemma 4.9. *Given a 3-partition problem instance (B, S) , we have*

$$A_{B,S} \text{ is satisfiable} \Leftrightarrow \exists \text{ complete 3-partition of } S \text{ (w.r.t. } B),$$

where $A_{B,S}$ is the symbolic heap given by Definition 4.8.

Proof. We establish each direction of the equivalence separately.

(\Leftarrow) Let $\{(k_{j_{i,1}}, k_{j_{i,2}}, k_{j_{i,3}}) \mid 1 \leq i \leq m\}$ be a complete 3-partition of S . We define a stack s by $s(d_1) = 0$ and, for all $1 \leq i \leq m$,

$$\begin{aligned} s(d_{i+1}) &= s(d_i) + B + 1, \\ s(a_{j_{i,1}}) &= s(d_i), \\ s(a_{j_{i,2}}) &= s(a_{j_{i,1}}) + k_{j_{i,1}}, \\ \text{and } s(a_{j_{i,3}}) &= s(a_{j_{i,2}}) + k_{j_{i,2}}. \end{aligned}$$

Notice that, using the equation $k_{j_{i,1}} + k_{j_{i,2}} + k_{j_{i,3}} = B$, we have

$$s(a_{j_{i,3}}) + k_{j_{i,3}} + 1 = s(d_i) + B + 1 = s(d_{i+1}).$$

Next we define a heap h (with arbitrarily chosen contents) by

$$\text{dom}(h) = \{s(d_1), s(d_1) + 1, \dots, s(d_{m+1})\}.$$

We claim that $s, h \models A_{B,S}$, as defined above.

First, we tackle the pure part. First, for each $1 \leq i \leq m$, we have $s \models d_{i+1} = d_i + B + 1$ by definition. Next, for each $1 \leq j \leq 3m$, we have by construction $a_j \geq 0 = d_1$. Finally, for all $1 \leq j \leq 3m$ we have, by construction and using the assumed bounds on each k ,

$$\begin{aligned} s(a_j) \leq s(a_{j_{m,3}}) &= s(d_m) + k_{j_{i,1}} + k_{j_{i,2}} \\ &\leq s(d_i) + B/2 + B/2 \\ &< s(d_i) + B + 1 \\ &= s(d_{m+1}). \end{aligned}$$

Thus indeed s satisfies the pure part of $A_{B,S}$.

Next, we check that s, h models the spatial part. We define $m + 1$ “heaplets” h_{d_i} by $\text{dom}(h_{d_i}) = \{s(d_i)\}$ for each $1 \leq i \leq m$, and $3m$ heaplets $h_{j_{i,\ell}}$ for each $1 \leq i \leq m$ and $\ell \in \{1, 2, 3\}$ by

$$\begin{aligned} \text{dom}(h_{j_{i,1}}) &= \{s(d_i) + 1, \dots, s(a_{j_{i,2}})\} \\ \text{dom}(h_{j_{i,2}}) &= \{s(a_{j_{i,2}}) + 1, \dots, s(a_{j_{i,3}})\} \\ \text{dom}(h_{j_{i,3}}) &= \{s(a_{j_{i,3}}) + 1, \dots, s(d_{i+1}) - 1\} \end{aligned}$$

(As before, the contents of these heaplets are irrelevant.)

By construction $s, h_{d_i} \models \text{array}(d_i, 0, 0)$ for each $1 \leq i \leq m + 1$. Similarly, for each $1 \leq i \leq m$ and $\ell \in \{1, 2, 3\}$ we have that $s, h_{j_{i,\ell}} \models \text{array}(a_{j_{i,\ell}}, 1, k_{j_{i,\ell}})$. Since each $j_{i,\ell}$ corresponds to a unique element in the sequence \mathcal{S} , this gives us the following $s, h_j \models \text{array}(a_j, 1, k_j)$ for each $1 \leq j \leq 3m$. We define h to be the \circ -composition of all our heaplets, i.e.,

$$h = \bigcirc_{1 \leq i \leq m+1} h_{d_i} \circ \bigcirc_{1 \leq j \leq 3m} h_j,$$

where the indexed “big circle” notation abbreviates a \circ -composition of heaps. To see that $s, h \models A_{B,S}$, we just need to show that h is well-defined, i.e., that all of our heaplets are non-overlapping. This holds by construction: for any $1 \leq i \leq m$ we have that h_{d_i} and $h_{d_{i+1}}$ are single cells separated by a contiguous gap of B cells, and the heaplets $h_{j_{i,1}}$, $h_{j_{i,2}}$ and $h_{j_{i,3}}$ are disjoint heaps occupying the gap between h_{d_i} and $h_{d_{i+1}}$. Thus $s, h \models A_{B,S}$ as required.

(\Rightarrow) Let s, h be a stack-heap pair satisfying $s, h \models A_{B,S}$. The spatial part of $A_{B,S}$ immediately yields a decomposition of h as

$$h = \bigcirc_{1 \leq i \leq m+1} h_{d_i} \circ \bigcirc_{1 \leq j \leq 3m} h_j,$$

where $\text{dom}(h_{d_i}) = \{s(d_i)\}$ for each $1 \leq i \leq m + 1$ and $\text{dom}(h_j) = \{s(a_j) + 1, \dots, s(a_j) + k_j\}$ for each $1 \leq j \leq 3m$; moreover, all of these “heaplets” are non-overlapping. In addition, the spatial part of $A_{B,S}$ yields $s(d_{i+1}) = s(d_i) + B + 1$ for all $1 \leq i \leq m + 1$, plus $s(d_1) \leq s(a_j)$ and $s(a_j) + k_j \leq d_{m+1}$ for all $1 \leq j \leq 3m$. This immediately implies that each heaplet h_j occupies a contiguous block of k_j cells between two successive single-cell heaplets h_{d_i} and $h_{d_{i+1}}$, which are themselves separated by a block of B cells. Moreover, because of the above equation $\sum_{j=1}^{3m} k_j = mB$, every such block of B cells must be *exactly covered* by h_j heaplets.

Now, we observe that, for each i , the block of B cells between h_{d_i} and $h_{d_{i+1}}$ must be covered by *precisely three* of our $3m$ heaplets: $h_{j_{i,1}}$, $h_{j_{i,2}}$ and $h_{j_{i,3}}$, say. This is due to the fact that $B/4 < k_j < B/2$ for each j : two heaplets are therefore insufficient to fill a gap of B cells, whereas four heaplets would occupy more than B cells (and would therefore overlap with each other or with h_{d_i} or $h_{d_{i+1}}$).

Therefore, we can define a 3-partition of \mathcal{S} by taking for each $1 \leq i \leq m$ the numbers $k_{i,1}$, $k_{i,2}$ and $k_{i,3}$ given by the sizes of the heaplets occupying the cells between h_{d_i} and $h_{d_{i+1}}$. It is immediate that $k_{i,1} + k_{i,2} + k_{i,3} = B$, as required. \square

C. Proofs of results in Section 5

Proposition 5.2. *If the biabduction problem (A, B) has a solution, then $\beta(A, B)$ is satisfiable.*

Proof. Let X, Y be a solution for (A, B) . This means that $A * X$ is satisfiable and that $A * X \models B * Y$. We conclude there exists a model that there is a model s, h such that $s, h \models A * X$ and $s, h \models B * Y$.

Since $s, h \models A * X$ this means that there is a subheap $h' \subseteq h$ such that $s, h' \models A$. Applying Lemma 4.5 to s, h' , we obtain that $s \models \gamma(A)$. The same reasoning on $s, h \models B * Y$ yields $s \models \gamma(B)$. It remains to show that

$$s \models \bigwedge_{j=1}^{\ell} \bigwedge_{i=1}^n (v_j < a_i \vee v_j > b_i) \wedge \bigwedge_{i=1}^{\ell} \bigwedge_{j=1}^k (t_i \neq v_j \vee u_i = w_j).$$

Suppose the left conjunct is false. Then, there are $j \in [1, \ell]$ and $i \in [1, n]$ for which $s \models a_i \leq v_j \leq b_i$. This means that the heap $h_j = \llbracket v_j \mapsto w_j \rrbracket^{s,h}$ is a *subheap* of the heap $h_i = \llbracket \text{array}(a_i, b_i) \rrbracket^{s,h}$. Let $\xi \in \text{Val}$ such that $\xi \neq s(w_j)$. It is easy to see that $s, h_i[s(v_j) \mapsto \xi] \models \text{array}(a_i, b_i)$ because the array predicate is insensitive to the values stored in the heap. This also means that $s, h[s(v_j) \mapsto \xi] \models A * X$. At the same time it is clear that $s, h_j[s(v_j) \mapsto \xi] \not\models v_j \mapsto w_j$. Therefore $s, h_j[s(v_j) \mapsto \xi] \not\models B * Y$, contradiction.

Suppose the right conjunct is false. Then, there are $i \in [1, \ell]$ and $j \in [1, k]$ such that $s \models t_i = v_j \wedge u_i \neq w_j$. Thus the heap $h_i = \llbracket t_i \mapsto u_i \rrbracket^{s,h}$ is well-defined, since $s, h \models A * X$. Similarly, the heap $h_j = \llbracket v_j \mapsto w_j \rrbracket^{s,h}$ is well-defined, because $s, h \models B * Y$. However, $s \models t_i = v_j$

meaning that $\text{dom}(h_i) = \text{dom}(h_j)$. On the other hand, $h_i(s(u_i)) \neq h_j(s(w_j))$ since $s \models u_i \neq w_j$. This is a contradiction, because both h_i and h_j are subheaps of h , but they have the same domain. This completes the proof. \square

Theorem 5.5. *If $\beta(A, B)$ is satisfiable, then there exists a solution seed Δ for the biabduction problem (A, B) .*

Proof. Supposing $s \models \beta(A, B)$, we define Δ as follows:

$$\Delta =_{\text{def}} \bigwedge_{\substack{e, f \in \mathcal{T}_{A,B} \\ s(e) < s(f)}} e < f \quad \wedge \quad \bigwedge_{\substack{e, f \in \mathcal{T}_{A,B} \\ s(e) = s(f)}} e = f.$$

We now check that Δ satisfies the conditions in Defn. 5.3.

Condition 3 holds because \leq is a total order over the set $\{s(e) \mid e \in \mathcal{T}_{A,B}\}$. Thus, the definition of Δ will introduce one of the atoms $f < e$, $e < f$ or $f = e$, for all $e, f \in \mathcal{T}_{A,B}$.

Condition 2 holds by construction.

Condition 1 requires that Δ is satisfiable. This follows by construction, as clearly s is a model of Δ .

Condition 1 also requires that $\Delta \models \beta(A, B)$. First we show $\Delta \models \gamma(A)$. Recall (Defn. 4.4) that, supposing A is written as in Defn. 5.3, we have

$$\gamma(A) = \Pi \wedge \bigwedge_{i \in [1, n+k]} \hat{a}_i \leq \hat{b}_i \wedge \bigwedge_{1 \leq i < j \leq n+k} (\hat{b}_i < \hat{a}_j) \vee (\hat{b}_j < \hat{a}_i)$$

where \hat{a}_i, \hat{b}_i are the endpoints of arrays in $[A]$, of which there are exactly $n + k$. Suppose π is a conjunct in Π . If π is of the form $t = u$ then, since $s \models \gamma(A)$ and thus $s \models \Pi$, we have $s \models t = u$; therefore by construction the conjunct $(t = u)$ appears in Δ and thus trivially $\Delta \models t = u$. The case for $t < u$ is similar. Suppose then that π is of the form $t \leq u$. Then, either $s(t) = s(u)$, in which case $(t = u)$ appears in Δ , or $s(t) < s(u)$ in which case $(t < u)$ appears in Δ . In both cases, $\Delta \models t \leq u$. Finally, if π is $t \neq u$ then it must be the case that either $(t < u)$ or $(u < t)$ appears in Δ , which again means that $\Delta \models t \neq u$. Therefore $\Delta \models \Pi$.

Next, let $i \in [1, n + k]$, and observe $\hat{a}_i, \hat{b}_i \in \mathcal{T}_{A,B}$. Since $s \models \gamma(A)$, we have $s(\hat{a}_i) \leq s(\hat{b}_i)$, meaning that either $s(\hat{a}_i) < s(\hat{b}_i)$ or $s(\hat{a}_i) = s(\hat{b}_i)$. Thus, by construction, either $(\hat{a}_i < \hat{b}_i)$ or $(\hat{a}_i = \hat{b}_i)$ is a conjunct of Δ , and in both cases $\Delta \models \hat{a}_i \leq \hat{b}_i$. Therefore, $\Delta \models \bigwedge_{i \in [1, n+k]} \hat{a}_i \leq \hat{b}_i$.

Finally, let $1 \leq i < j \leq n$, and observe $\hat{a}_i, \hat{b}_i, \hat{a}_j, \hat{b}_j$ are all in $\mathcal{T}_{A,B}$. Since $s \models \gamma(A)$ by assumption, we have $s \models (\hat{b}_i < \hat{a}_j) \vee (\hat{b}_j < \hat{a}_i)$, meaning that either $s(\hat{b}_i) < s(\hat{a}_j)$ or $s(\hat{b}_j) < s(\hat{a}_i)$. Thus either $(\hat{b}_i < \hat{a}_j)$ or $(\hat{b}_j < \hat{a}_i)$ is a conjunct of Δ , so $\Delta \models (\hat{b}_i < \hat{a}_j) \vee (\hat{b}_j < \hat{a}_i)$. This gives us $\Delta \models \bigwedge_{1 \leq i < j \leq n} (\hat{b}_i < \hat{a}_j) \vee (\hat{b}_j < \hat{a}_i)$. Putting everything together, we get $\Delta \models \gamma(A)$. The argument that $\Delta \models \gamma(B)$ is identical.

Next, we show $\Delta \models \bigwedge_{j=1}^{\ell} \bigwedge_{i=1}^n (v_j < a_i \vee v_j > b_i)$. We know that $s \models v_j < a_i \vee v_j > b_i$ for all $j \in [1, \ell]$ and $i \in [1, n]$. Thus $s \models v_j < a_i$ or $s \models v_j > b_i$, meaning $s(v_j) < s(a_i)$ or $s(v_j) > s(b_i)$. By the fact

$v_j, a_i, b_i \in \mathcal{T}_{A,B}$ and the definition of Δ we know that one of $(v_j < a_i)$ or $(b_i < v_j)$ is a conjunct of Δ . Thus $\Delta \models (v_j < a_i) \vee (b_i < v_j)$ and we are done.

Finally, we show $\Delta \models \bigwedge_{i=1}^{\ell} \bigwedge_{j=1}^k (t_i \neq v_j \vee u_i = w_j)$. Again, we know that $s \models t_i \neq v_j \vee u_i = w_j$ for all $i \in [1, \ell]$ and $j \in [1, k]$. There are two cases: $s \models u_i = w_j$ or $s \models t_i \neq v_j$. In the first case, $\Delta \models u_i = w_j$ by construction. In the latter case, there are two further subcases, namely $s \models t_i < v_j$ or $s \models t_i > v_j$ and it can be easily seen that in both of these, $\Delta \models t_i \neq v_j$. This completes the proof. \square

Proposition 5.7. *Let (A, B) be a biabduction problem of the form shown in Defn. 5.3. Let Δ be a solution seed and terms $e, f \in \mathcal{T}_{A,B}$. The call $\text{arrcov}_{A,\Delta}(e, f)$:*

1. *always terminates, issuing up to $n + k$ recursive calls;*
2. *returns a formula (for some $q \in \mathbb{N}$ and sets $I, J \subseteq \mathbb{N}$)*

$$\bigwedge_{i \in I} a_i = a'_i + 1 \wedge \bigwedge_{i \in J} t_i = t'_i + 1 : \bigstar_{i=1}^q \text{array}(l_i, r_i)$$

where for all $i \in [1, q]$, $l_i \in \mathcal{T}_{A,B}$;

3. *for every $i \in [1, q]$, $\hat{\Delta} \models e \leq l_i \leq r_i \leq f$;*
4. *for every $i \in [1, q - 1]$, $\hat{\Delta} \models r_i < l_{i+1}$.*

Proof. First, note that there are exactly $n + k$ arrays in $[A]$, hence the upper limit of \bigstar in line 2.

Termination follows from the fact that $\text{arrcov}_{A,\Delta}(e, f)$ either terminates immediately when $f <_{\Delta} e$, or recurses with calls of the form $\text{arrcov}_{A,\Delta}(b_{i_j} + 1, f)$, where the sequence b_{i_j} is $<_{\Delta}$ -increasing, thus terminating at the first index i_j such that $f <_{\Delta} b_{i_j} + 1$. There can be up to $n + k$ such calls.

To show items 2 and 3, we examine each section of the algorithm, and argue by induction over the recursion depth.

If $f <_{\Delta} e$ then the algorithm terminates at line 4, returning emp, a result of the required form.

Otherwise, $e \leq_{\Delta} f$ (by Lemma 5.4). If $a_i \leq_{\Delta} e \leq_{\Delta} b_i$ for some $i \in [1, n + k]$ (line 6), then the recursive call $\text{arrcov}_{A,\Delta}(b_i + 1, f)$ is issued. Since $e \leq_{\Delta} b_i$, we know that $e <_{\Delta} b_i + 1$.

Otherwise, there is no i such that $a_i \leq_{\Delta} e \leq_{\Delta} b_i$. If the set E is empty (line 10), then the algorithm terminates returning a result that is, trivially, of the required form.

Otherwise, there is a minimal element in E , namely a_i . In this case, a recursive call $\text{arrcov}_{A,\Delta}(b_i + 1, f)$ is issued, with $e < b_i + 1$. By the inductive hypothesis and the lifting of \bigstar to symbolic heaps, we obtain a result of the required form.

That for every $i \in [1, q]$, $\hat{\Delta} \models e \leq l_i \leq r_i \leq f$ follows by inspecting the array constructors used in the code. In particular, $\text{array}(e, f)$ (line 11) trivially provides the required condition (note that $e \leq_{\Delta} f$ by line 3). For $\text{array}(e, \hat{a}'_i)$ at line 14, observe that $e <_{\Delta} \hat{a}_i$ holds by the definition of E at line 9. Moreover, $\Theta_X \models \hat{a}_i = \hat{a}'_i + 1$, thus $\hat{\Delta} \models e \leq \hat{a}'_i$.

Line 14 also guarantees item (4): this is the only place in the code where multiple arrays may be returned, and we

clearly have $\hat{\Delta} \models \hat{a}'_i < \hat{b}_i + 1$, which, combined with item (3) completes the proof. \square

We will use the expression $\langle \nu \mapsto \xi \rangle$, where $\nu, \xi \in \mathbb{N}$, to denote the heap h such that $\text{dom}(h) = \{\nu\}$ and $h(\nu) = \xi$.

Lemma 5.8. *Let (A, B) be a biabduction instance, Δ a solution seed and X as in Defn. 5.6. Then, $A * X$ is satisfiable.*

Proof. By Defn. 5.3 we know there is a stack \hat{s} such that $\hat{s} \models \Delta$. We define a stack s_X that correctly assigns values to primed terms, as added by arrcov .

$$s_X(e) =_{\text{def}} \begin{cases} \hat{s}(e) & e \in \mathcal{T}_{A,B} \\ \hat{s}(a_i) - 1 & e \equiv a'_i \in FV(X), \text{ for } i \in [1, n] \\ \hat{s}(t_i) - 1 & e \equiv t'_i \in FV(X), \text{ for } i \in [1, k] \end{cases}$$

Observe that the variables a'_i and t'_i are fresh in Δ and appear at most once in Θ_X (this is due to Prop. 5.7). We must show that s_X is well defined, i.e., there is no term $a'_i \in FV(X)$ such that $\hat{s}(a_i) = 0$, and equally for terms t'_i .

Suppose there is such an a'_i . Then there must be some $j \in [1, n]$ for which the call $\text{arrcov}_{A,\Delta}(a_j, b_j)$ reaches line 14 which introduces the constraint $a'_i + 1 = a_i$. But in that branch, $e <_{\Delta} a_i$. Thus, it must be that $\hat{s}(a_i) \neq 0$. The same argument applies to primed terms t'_i , meaning s_X is well defined.

In addition, s_X agrees with s on all variables in Δ , thus $s_X \models \Delta \wedge \Theta_X$. A similar argument constructs another stack s such that $s \models \Delta \wedge \Theta_X \wedge \Theta_Y$.

We now define several heaps.

$$\mathcal{A}_i^{\text{pto}} =_{\text{def}} \langle s(t_i) \mapsto s(u_i) \rangle \quad \forall i \in [1, k]$$

$$\mathcal{A}_i^{\text{arr}} =_{\text{def}} \bigcirc_{\nu=s(a_i)}^{s(b_i)} \langle \nu \mapsto 0 \rangle \quad \forall i \in [1, n]$$

$$\mathcal{A} =_{\text{def}} \bigcirc_{i=1}^k \mathcal{A}_i^{\text{pto}} \circ \bigcirc_{i=1}^n \mathcal{A}_i^{\text{arr}}$$

It is easy to see that, as $s \models \Delta$ and $\Delta \models \gamma(A)$, all heaps $\mathcal{A}_i^{\text{pto}}$ and $\mathcal{A}_i^{\text{arr}}$ are well-defined and disjoint. As a consequence, \mathcal{A} is well-defined. By construction, $s, \mathcal{A} \models A$. We continue by defining heaps $\mathcal{X}_i^{\text{pto}}$ and $\mathcal{X}_i^{\text{arr}}$.

$$\mathcal{X}_i^{\text{pto}} =_{\text{def}} \begin{cases} e & s(v_i) \in \text{dom}(\mathcal{A}) \\ \langle s(v_i) \mapsto s(w_i) \rangle & \text{otherwise} \end{cases} \quad \forall i \in [1, \ell]$$

$$\mathcal{X}_i^{\text{arr}} =_{\text{def}} \bigcirc_{\nu \in [s(c_i), s(d_i)] \setminus \text{dom}(\mathcal{A})} \langle \nu \mapsto 0 \rangle \quad \forall i \in [1, m]$$

$$h =_{\text{def}} \mathcal{A} \circ \bigcirc_{i=1}^{\ell} \mathcal{X}_i^{\text{pto}} \circ \bigcirc_{i=1}^m \mathcal{X}_i^{\text{arr}}$$

First, observe that, by construction, $\mathcal{X}_i^{\text{arr}} \# \mathcal{A}$ ($i \in [1, m]$) and $\mathcal{X}_i^{\text{pto}} \# \mathcal{A}$ ($i \in [1, \ell]$).

Also, note that $\mathcal{X}_i^{\text{pto}} \# \mathcal{X}_j^{\text{pto}}$ for $i \neq j \in [1, \ell]$ since otherwise $s(v_i) = s(v_j)$ which contradicts $s \models \gamma(B)$,

deriving from $s \models \Delta$ and $\Delta \models \gamma(B)$. Equally, $\mathcal{X}_i^{\text{arr}} \# \mathcal{X}_j^{\text{arr}}$ for $i \neq j \in [1, m]$ by a similar argument. Finally, $\mathcal{X}_i^{\text{pto}} \# \mathcal{X}_j^{\text{arr}}$ for $i \in [1, \ell]$ and $j \in [1, m]$ as otherwise $s(c_j) \leq s(v_i) \leq s(d_j)$, contradicting again $s \models \gamma(B)$. Thus, h is well-defined.

It is not hard to verify that for each $i \in [1, \ell]$,

$$s, \mathcal{X}_i^{\text{pto}} \models \llbracket \text{ptocov}_{B,\Delta}(v_i, w_i) \rrbracket^{s,h}.$$

We show the last obligation, i.e., that for $i \in [1, m]$

$$s, \mathcal{X}_i^{\text{arr}} \models \llbracket \text{arrcov}_{B,\Delta}(c_i, d_i) \rrbracket^{s,h}.$$

Suppose the opposite. Due to the form of the result returned by $\text{arrcov}_{B,\Delta}(c_i, d_i)$ as guaranteed by Prop. 5.7, this means there must exist some address ν such that either $\nu \in \text{dom}(\mathcal{X}_i^{\text{arr}}) \setminus \text{dom}(\llbracket \text{arrcov}_{B,\Delta}(c_i, d_i) \rrbracket^{s,h})$, or conversely, $\nu \in \text{dom}(\llbracket \text{arrcov}_{B,\Delta}(c_i, d_i) \rrbracket^{s,h}) \setminus \text{dom}(\mathcal{X}_i^{\text{arr}})$.

In the first case, there must be some $\text{array}(e, f)$ returned by $\text{arrcov}_{B,\Delta}(c_i, d_i)$ such that $[s(e), s(f)] \not\subseteq \text{dom}(\mathcal{X}_i^{\text{arr}})$. We know, however, that $[s(e), s(f)] \subseteq [s(c_i), s(d_i)]$, from Prop. 5.7. But then $\nu \in [s(c_i), s(d_i)]$ thus, by the definition of $\mathcal{X}_i^{\text{arr}}$, it must be that $\nu \in \text{dom}(\mathcal{A})$. This contradicts both of the cases where an array is returned by arrcov (lines 11 and 14).

In the second case, there is some address $\nu \in \text{dom}(\mathcal{X}_i^{\text{arr}})$ such that there is no $\text{array}(e, f)$ returned by $\text{arrcov}_{B,\Delta}(c_i, d_i)$, such that $\nu \in [s(e), s(f)]$. Again, by assumption we have $\nu \in [s(c_i), s(d_i)]$. However, it can be verified by inspecting arrcov that if $\nu \in [s(c_i), s(d_i)]$ and there is no $\text{array}(e, f)$ in the result, this is because $\nu \in \text{dom}(\mathcal{A})$, contradicting the fact that $\mathcal{X}_i^{\text{arr}} \# \mathcal{A}$. This completes the proof. \square

Lemma 5.10. *All elements of $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$ are well-defined, in the sense that there exist such (unique) heaps.*

Proof. Uniqueness follows by the above observation that all quantifier-free formulas in ASL are precise. Here we show existence.

Suppose $\mathcal{B}_i^{\text{pto}}$ is not well-defined, meaning $s(v_i) \notin \text{dom}(h)$, or that $h(s(v_i)) \neq s(w_i)$. In the first case, it must be that $\text{ptocov}_{A,\Delta}(v_i, w_i) = \text{emp}$ (otherwise, by construction, $s(v_i) \in \text{dom}(\llbracket X \rrbracket^{s,h})$). But this happens exactly when $s(v_i) \in \text{dom}(\llbracket A \rrbracket^{s,h}) \subseteq \text{dom}(h)$, contradiction. In the second case, suppose $s(v_i) \notin \text{dom}(\llbracket A \rrbracket^{s,h})$. This means $\text{ptocov}_{A,\Delta}(v_i, w_i) = v_i \mapsto w_i$ which by construction guarantees $h(s(v_i)) = s(w_i)$. Finally, suppose $s(v_i) \in \text{dom}(\llbracket A \rrbracket^{s,h})$. Either there is an $\text{array}(a_j, b_j)$ such that $s(v_i) \in [s(a_i), s(b_i)]$, or there is $t_j \mapsto w_j$ such that $s(v_i) = s(t_j)$. The first possibility contradicts the second conjunct of $\beta(A, B)$ and the second possibility the third conjunct.

Suppose $\mathcal{B}_i^{\text{arr}}$ is not well-defined, meaning $[s(c_i), s(d_i)] \not\subseteq \text{dom}(h)$. In other words, there is $\nu \in [s(c_i), s(d_i)]$, but $\nu \notin \text{dom}(h)$. Clearly, $\nu \notin \text{dom}(\llbracket A \rrbracket^{s,h})$. By inspecting arrcov , however, we can conclude that there must be

some $\text{array}(e, f)$ returned by $\text{arrcov}_{A,\Delta}(c_i, d_i)$ such that $\nu \in [s(e), s(f)]$. This means $\nu \in \text{dom}(\llbracket X \rrbracket^{s,h})$, contradiction.

Suppose $\mathcal{Y}_i^{\text{pto}}$ is not well defined. This must mean $\mathcal{Y}_i^{\text{pto}} \neq e$, because trivially $e \subseteq h$. For this to happen, $\text{ptocov}_{B,\Delta}(t_i, u_i)$ must return $t_i \mapsto u_i$, and $s(t_i) \notin \text{dom}(h)$. But by assumption, $s, h \models A * X$, therefore $s(t_i) \in \text{dom}(h)$, contradiction.

Suppose $\mathcal{Y}_i^{\text{arr}}$ is not well-defined. Thus, there is some $\text{array}(e, f)$ returned by $\text{arrcov}_{B,\Delta}(a_i, b_i)$ such that $[s(e), s(f)] \not\subseteq \text{dom}(h)$. However, we know $s(a_i) \leq s(e) \leq s(f) \leq s(b_i)$ from Prop. 5.7. Also, by assumption, $s, h \models A * X$ thus $[s(a_i), s(b_i)] \subseteq \text{dom}(h)$, contradiction. \square

Lemma 5.10. 1. For any sequence of heaps \mathcal{S} of $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$ and any distinct $i, j \in [1, |\mathcal{S}|]$, $\mathcal{S}_i \# \mathcal{S}_j$.
2. For any two distinct sequences of heaps \mathcal{S}, \mathcal{T} of $\mathcal{B}^{\text{arr}}, \mathcal{B}^{\text{pto}}, \mathcal{Y}^{\text{arr}}, \mathcal{Y}^{\text{pto}}$, and any $i \in [1, |\mathcal{S}|]$, $j \in [1, |\mathcal{T}|]$, $\mathcal{S}_i \# \mathcal{T}_j$.

Proof. For \mathcal{B}^{arr} and \mathcal{B}^{pto} , this follows from the fact that $s \models \gamma(B)$, ensuring the separation of arrays and \mapsto formulas in B .

We show that for any $i \neq j$, $\mathcal{Y}_i^{\text{pto}}$ and $\mathcal{Y}_j^{\text{pto}}$ are disjoint. By inspecting $\text{ptocov}_{B,\Delta}(t_i, u_i)$, we see that $\mathcal{Y}_i^{\text{pto}}$ is either \emptyset , or $\{s(t_i)\}$. If either of $\mathcal{Y}_i^{\text{pto}}, \mathcal{Y}_j^{\text{pto}}$ is the empty heap e , for $i \neq j \in [1, k]$, then clearly $\mathcal{Y}_i^{\text{pto}} \# \mathcal{Y}_j^{\text{pto}}$. If both are non-empty, then their domains are $\{s(t_i)\}, \{s(t_j)\}$ (line 8). But, by assumption, $s \models \gamma(A)$ which guarantees $s(t_i) \neq s(t_j)$.

For any $i \neq j$, $\mathcal{Y}_i^{\text{arr}}$ and $\mathcal{Y}_j^{\text{arr}}$ are disjoint, because item (3) of Prop. 5.7 means that $\mathcal{Y}_i^{\text{arr}} \subseteq \llbracket \text{array}(a_i, b_i) \rrbracket^{s,h}$, for $i \in [1, n]$. But $\llbracket \text{array}(a_i, b_i) \rrbracket^{s,h} \# \llbracket \text{array}(a_j, b_j) \rrbracket^{s,h}$ for $i \neq j$, due to $s \models \gamma(A)$.

We also need to show that for any pair of heaps from any two of these sequences, the heaps are disjoint. In the case of heaps $\mathcal{B}_i^{\text{arr}}, \mathcal{B}_j^{\text{pto}}, \mathcal{B}_i^{\text{arr}} \# \mathcal{B}_j^{\text{pto}}$ follows again from the assumption that $s \models \gamma(B)$.

Suppose it is not the case that $\mathcal{B}_i^{\text{pto}} \# \mathcal{Y}_j^{\text{pto}}$. As argued previously, it must be that $\text{dom}(\mathcal{Y}_j^{\text{pto}}) = \{s(t_j)\}$. At the same time, $\text{dom}(\mathcal{B}_i^{\text{pto}}) = \{s(v_i)\}$, meaning that $s(t_j) = s(v_i)$. Since, $t_j, v_i \in \mathcal{T}_{A,B}$, it must be that $t_j =_{\Delta} v_i$. But then, $\text{ptocov}_{B,\Delta}(t_i, u_i)$ would return emp (line 4), contradiction.

Suppose it is not the case that $\mathcal{B}_i^{\text{arr}} \# \mathcal{Y}_j^{\text{pto}}$. Again, this means $\text{dom}(\mathcal{Y}_j^{\text{pto}}) = \{s(t_j)\}$. Since $\text{dom}(\mathcal{B}_i^{\text{arr}}) = [s(c_i), s(d_i)]$, we conclude that $c_i \leq_{\Delta} t_j \leq_{\Delta} d_i$. We again have a contradiction, as in this case $\text{ptocov}_{B,\Delta}(t_i, u_i)$ would return emp (line 7).

Next, suppose it does not hold that $\mathcal{Y}_i^{\text{arr}} \# \mathcal{Y}_j^{\text{pto}}$. As above, this means $\text{dom}(\mathcal{Y}_j^{\text{pto}}) = \{s(t_j)\}$. In addition, there must be some $\text{array}(e, f)$ returned by $\text{arrcov}_{B,\Delta}(a_i, b_i)$ such that $\llbracket \text{array}(e, f) \rrbracket^{s,h} \# \mathcal{Y}_j^{\text{pto}}$ does not hold, meaning that $\hat{\Delta} \models e \leq t_j \leq f$. By Prop. 5.7 we know that $\hat{\Delta} \models a_i \leq e \leq f \leq b_i$ thus $a_i \leq_{\Delta} t_j \leq_{\Delta} b_i$. This contradicts the assumption $s \models \gamma(A)$.

Now suppose it is not the case that $\mathcal{B}_i^{\text{pto}} \# \mathcal{Y}_j^{\text{arr}}$. Note that $\text{dom}(\mathcal{B}_i^{\text{pto}}) = \{s(v_i)\}$. As above, there must be some $\text{array}(e, f)$ in the result of $\text{arrcov}_{B,\Delta}(a_j, b_j)$ such that $s(e) \leq s(v_i) \leq s(f)$, thus $a_j \leq_{\Delta} v_i \leq_{\Delta} b_j$, contradicting the second conjunct of Defn. 5.1.

Finally, we need to show that $\mathcal{B}_i^{\text{arr}}, \mathcal{Y}_j^{\text{arr}}$ are disjoint. Suppose the contrary. This means that there is an $\text{array}(e, f)$ in the result of $\text{arrcov}_{B,\Delta}(a_j, b_j)$ such that $c_i \leq_{\Delta} e \leq_{\Delta} d_i$ or $c_i \leq_{\Delta} f \leq_{\Delta} d_i$. We inspect the return statements of arrcov where an array is constructed. At line 11, the array constructed is $\text{array}(e, f)$. At this point there is no $q \in [1, m + \ell]$ such that $\hat{c}_q \leq_{\Delta} e \leq_{\Delta} \hat{d}_q$ (because of line 6) or $\hat{c}_q \leq_{\Delta} f \leq_{\Delta} \hat{d}_q$ (because of line 10). At line 14, the array constructed is $\text{array}(e, \hat{c}_q')$ for some $q \in [1, m + \ell]$ such that $e <_{\Delta} \hat{c}_q \leq_{\Delta} f$, and \hat{c}_q is the $<_{\Delta}$ -minimal such array endpoint. Clearly, there is no r such that $\hat{c}_r \leq_{\Delta} e \leq_{\Delta} \hat{d}_r$ (again because of line 6). Thus we need only show that there is no r such that $\hat{c}_r \leq_{\Delta} \hat{c}_q' \leq_{\Delta} \hat{d}_r$. But this is provided directly by the fact that $s \models \gamma(B)$. \square

Lemma 5.10.

$$\text{dom}(h) \subseteq \bigcup_{i=1}^m \mathcal{B}_i^{\text{arr}} \cup \bigcup_{i=1}^{\ell} \mathcal{B}_i^{\text{pto}} \cup \bigcup_{i=1}^n \mathcal{Y}_i^{\text{arr}} \cup \bigcup_{i=1}^k \mathcal{Y}_i^{\text{pto}}$$

Proof. We show that for all atomic formulas σ of F_{A*X} there is a set of heaps \mathcal{H} from the above sequences such that $\llbracket \sigma \rrbracket^{s,h} \subseteq \bigcirc \mathcal{H}$.

Recall that $F_{A*X} = F_A * F_X$ and that

$$F_A = \bigstar_{i=1}^n \text{array}(a_i, b_i) * \bigstar_{i=1}^k t_i \mapsto u_i$$

$$F_X = \bigstar_{i=1}^m \text{arrcov}_{A,\Delta}(c_i, d_i) * \bigstar_{i=1}^{\ell} \text{ptocov}_{A,\Delta}(v_i, w_i)$$

We deal with the four subcases depending on the provenance of σ .

Let $\sigma \equiv t_i \mapsto u_i$ for some $i \in [1, k]$. The call $\text{ptocov}_{B,\Delta}(t_i, u_i)$ will return $t_i \mapsto u_i$ or emp . In the former case $\llbracket \sigma \rrbracket^{s,h} = \mathcal{Y}_i^{\text{pto}}$. Otherwise, there is some $j \in [1, \ell]$ such that $t_i =_{\Delta} v_j$ or there is $j \in [1, m]$ such that $c_j \leq_{\Delta} t_i \leq_{\Delta} d_j$. In the first case, $\llbracket \sigma \rrbracket^{s,h} = \mathcal{B}_j^{\text{pto}}$ and in the second, $\llbracket \sigma \rrbracket^{s,h} \subseteq \mathcal{B}_j^{\text{arr}}$.

Let $\sigma \equiv e \mapsto f$, returned by $\text{ptocov}_{A,\Delta}(v_i, w_i)$ for $i \in [1, \ell]$. By inspecting ptocov it can be seen that, necessarily, $e \equiv v_i$. But then, trivially, $\llbracket \sigma \rrbracket^{s,h} = \mathcal{B}_i^{\text{pto}}$ and we are done.

Let $\sigma \equiv \text{array}(e, f)$, returned by $\text{arrcov}_{A,\Delta}(c_i, d_i)$ for $i \in [1, m]$. By Prop. 5.7 we know that $c_i \leq_{\Delta} e \leq_{\Delta} f \leq_{\Delta} d_i$, meaning that $\llbracket \sigma \rrbracket^{s,h} \subseteq \mathcal{B}_i^{\text{arr}}$.

Let $\sigma \equiv \text{array}(a_i, b_i)$ for some $i \in [1, n]$, and let $\mathcal{H}_B = \bigcirc_{i=1}^{\ell} \mathcal{B}_i^{\text{pto}} \circ \bigcirc_{i=1}^m \mathcal{B}_i^{\text{arr}}$. We argue that $\llbracket \sigma \rrbracket^{s,h} \subseteq \mathcal{Y}_i^{\text{arr}} \circ \mathcal{H}_B$. We do this by proving that for any $e, f \in \mathcal{T}_{A,B}$ such that $a_i \leq_{\Delta} e \leq_{\Delta} f \leq_{\Delta} b_i$, $\llbracket \text{array}(e, f) \rrbracket^{s,h} \subseteq \mathcal{Y}_i^{\text{arr}} \circ \mathcal{H}_B$, and we do this by induction over the recursion depth.

If the depth is zero, then there is no array in $[B]$ covering e and there is no array covering f (line 11). Thus, $\text{arrcov}_{B,\Delta}(e, f) = \text{array}(e, f)$, therefore trivially $\llbracket \text{arrcov}_{B,\Delta}(e, f) \rrbracket^{s,h} \subseteq \llbracket \text{array}(a_i, b_i) \rrbracket^{s,h} = \mathcal{Y}_i^{\text{arr}}$.

If the depth is non-zero, either there is $\text{array}(\hat{c}_j, \hat{d}_j)$ such that $s(e) \in [s(\hat{c}_j), s(\hat{d}_j)]$ (line 6) or there is no such array, but there is a (left-most) $\text{array}(\hat{c}_j, \hat{d}_j)$ covering f (line 13).

In the first case, $\text{arrcov}_{B,\Delta}(e, f) = \text{arrcov}_{B,\Delta}(\hat{d}_j + 1, f)$, where $a_i \leq_\Delta e \leq_\Delta \hat{d}_j <_\Delta \hat{d}_j + 1$. If $f <_\Delta \hat{d}_j + 1$ then the call $\text{arrcov}_{B,\Delta}(\hat{d}_j + 1, f) = \text{emp}$ and we are done. Otherwise, the inductive hypothesis applies and we get $\text{arrcov}_{B,\Delta}(e, f) = \text{arrcov}_{B,\Delta}(\hat{d}_j + 1, f) \subseteq \mathcal{Y}_i^{\text{arr}} \circ \mathcal{H}_B$.

In the second case, $\text{arrcov}_{B,\Delta}(e, f)$ is equal to

$$\hat{c}'_j + 1 = \hat{c}_j : \text{array}(e, \hat{c}'_j) * \text{arrcov}_{B,\Delta}(\hat{d}_j + 1, f).$$

This means that $\llbracket \text{arrcov}_{B,\Delta}(e, f) \rrbracket^{s,h}$ is equal to

$$\llbracket \text{array}(e, \hat{c}'_j) \rrbracket^{s,h} \cup \llbracket \text{arrcov}_{B,\Delta}(\hat{d}_j + 1, f) \rrbracket^{s,h}$$

Clearly, $\llbracket \text{array}(e, \hat{c}'_j) \rrbracket^{s,h} \subseteq \mathcal{Y}_i^{\text{arr}}$ so we need to show the same for $\llbracket \text{arrcov}_{B,\Delta}(\hat{d}_j + 1, f) \rrbracket^{s,h}$. This follows by an identical argument to the previous case, via the inductive hypothesis. This completes the proof. \square

Lemma 5.13. *Let (A, B) be a biabduction instance and Δ a formula satisfying Conditions 2 and 3 of Defn. 5.3. Let $\Gamma = \bigwedge \bigvee \pi$ be a formula where π is of the form $t < u$ or $t = u$ and $t, u \in \mathcal{T}_{A,B}$. Then, checking $\Delta \models \Gamma$ is in PTIME.*

Proof. Let $\Delta = \bigwedge_{i \in I} \delta_i$. First, we assume that $J = K = \{1\}$, i.e., that the query in question is simply $\Delta \models \pi$ for a single atomic formula π . Let π be of the form $t < u$. If there exists $i \in I$ such that $\delta_i \equiv \pi$ then clearly $\Delta \models \pi$, as Δ is a conjunction. We return “yes”.

If there is no $i \in I$ such that $\delta_i \equiv \pi$ then by the assumption that $t, u \in \mathcal{T}_{A,B}$ and Condition 3 of Defn. 5.3 we have that $t = u$ or $u < t$ is a conjunct of Δ . In both cases it is clear that $\Delta \not\models \pi$ (again because Δ is a conjunction) and we return “no”.

The case where π is of the form $t = u$ is almost identical. Thus we can answer queries of the form $\Delta \models \pi$ in time linear in $|\Delta|$.

Suppose now that $\Gamma = \bigvee_{k \in K} \pi_k$, that is, I is a singleton. We issue all possible queries of the form $\Delta \models \pi_k$ for $k \in K$. If any of these queries reports “yes” then we report “yes”. Otherwise, due to the completeness of checking these queries we have that $\Delta \models \bigwedge_{k \in K} \neg \pi_k$ and we report “no”. Therefore, queries of the form $\Delta \models \bigvee_{k \in K} \pi_k$ can be checked in time $|\Delta| \cdot |K|$.

Finally, suppose $\Gamma = \bigwedge_{j \in J} \bigvee_{k \in K} \pi_{j,k}$. We issue $|J|$ queries of the form $\Delta \models \bigvee_{k \in K} \pi_{j,k}$ for each $j \in J$. If all queries receive positive answers then clearly $\Delta \models \Gamma$ and we return “yes”. Otherwise there is $j \in J$ such that the query $\Delta \models \bigvee_{k \in K} \pi_{j,k}$ received a negative answer, meaning that

$\Delta \not\models \bigvee_{k \in K} \pi_{j,k}$. Thus, as Γ is a conjunction at the top-level, it is clear that $\Delta \not\models \Gamma$ and we report “no”. This last step can take up to $|\Delta| \cdot |K| \cdot |J|$ time. \square

Proposition 5.14. *Deciding if there is a solution for a biabduction problem (A, B) , and constructing it if it exists, can be done in NP.*

Proof. We outline a non-deterministic algorithm that runs in polynomial time in the size of the input (A, B) .

First, we guess a set $T \subseteq \mathcal{T}_{A,B} \times \{<, =\} \times \mathcal{T}_{A,B}$. Next, we guess an assignment of values s to the variables in $\mathcal{T}_{A,B}$. We limit the range of the assignment to naturals bounded by some B which is exponential in $|\mathcal{T}_{A,B}|$ (representable in polynomial space and guessable in non-deterministic polynomial time). The precise definition of B is not relevant here, and is given in [34, Theorem 6].

We then convert the set T (of quadratic size in $|\mathcal{T}_{A,B}|$) into a formula Δ in the obvious way. The resulting Δ automatically satisfies Condition 2 of Defn. 5.3. Condition 3 of Defn. 5.3 is checkable in quadratic time by a nested loop over pairs of terms from $\mathcal{T}_{A,B}$, scanning Δ in each iteration. The formula $\beta(A, B)$ can be split into a fixed number of formulas of the form $\bigwedge \bigvee \pi$, given that $t \leq u$ is equivalent to $t < u \vee t = u$. Thus, $\Delta \models \beta(A, B)$ can be checked in polynomial time due to Lemma 5.13. Finally, we check that Δ is satisfiable by checking whether $s \models \Delta$. This step can be done in polynomial time and is complete by [34, Theorem 6]. If all checks pass, then Δ is a solution seed.

We now apply Defn. 5.6 on Δ and obtain the formulas X and Y . By Prop. 5.7, each call $\text{arrcov}_{A,\Delta}(c_j, d_j)$ issues at most $n + k$ recursive calls. The work done in each call is clearly doable in polynomial time (cf. Lemma 5.13), thus completing the proof. \square

Proposition 5.18. *Let A be quantifier-free, and let B be such that no variable appearing in the RHS of a \mapsto formula is existentially bound. Then an instance (A, B) of the biabduction problem for ASL has a solution if and only if $(A, \text{qf}(B))$ has a solution.*

Proof. Let $B = \exists \mathbf{z}. Q$, where $Q = \text{qf}(B)$ is quantifier-free. We tackle each direction of the equivalence separately.

(\Leftarrow) Let (X, Y) be a solution for (A, Q) . We claim that (X, Y) is also a solution for $(A, \exists \mathbf{z}. Q)$. To see this, observe that by assumption $A * X$ is satisfiable and $A * X \models Q * Y$. Since trivially $Q \models \exists \mathbf{z}. Q$, we easily have $Q * Y \models (\exists \mathbf{z}. Q) * Y$ and so $A * X \models (\exists \mathbf{z}. Q) * Y$ as required.

(\Rightarrow) Let (X, Y) be a solution for $(A, \exists \mathbf{z}. Q)$. That is, $A * X$ is satisfiable and $A * X \models (\exists \mathbf{z}. Q) * Y$. Since the free variables in Y are disjoint from \mathbf{z} , this can be rewritten as $A * X \models \exists \mathbf{z}. (Q * Y)$. Now, by assumption there is a stack-heap pair (s, h) such that $s, h \models A * X$. Furthermore, h is clearly independent of the data values stored in the arrays

in $A * X$. Thus we may choose h such that $h(x) \neq s(w)$ for all formulas of the form $v \mapsto w$ occurring in $Q * Y$, and for all x such that $\text{array}(a, b)$ occurs in $A * X$ and $s(a) \leq s(x) \leq s(b)$.

Now, since $A * X \models (\exists \mathbf{z}. Q) * Y$, we get $s, h \models \exists \mathbf{z}. (Q * Y)$, meaning that $s[\mathbf{z} \mapsto \mathbf{m}], h \models Q * Y$ for some \mathbf{m} . We write $s' = s[\mathbf{z} \mapsto \mathbf{m}]$, and define the following extension of the symbolic heap X :

$$X' =_{\text{def}} \left(\bigwedge_{x \in FV(A, X, Q, Y)} x = s'(x) \right) * X$$

We claim that (X', Y) is then a solution for (A, Q) . First, since $s, h \models A * X$ but the variables \mathbf{z} do not occur in $A * X$ by assumption, we also have $s', h \models A * X$. Clearly, we also have $s' \models x = s'(x)$ for any x , and so $s', h \models A * X'$. Hence $A * X'$ is satisfiable.

It remains to show that $A * X' \models Q * Y$. Supposing $s'', h' \models A * X'$, we require to prove $s'', h' \models Q * Y$. By construction of X' , the stack s'' agrees with s' on all variables occurring in A, X', Q and Y , so in fact we have $s', h' \models A * X'$ and require to prove $s', h' \models Q * Y$.

Now, since $s', h \models A * X'$ and $s', h' \models A * X'$, we have $\text{dom}(h') = \text{dom}(h)$ by Lemma 3.3. Since $s', h \models Q * Y$, it is then easy to see that s', h' satisfies all pure formulas and all array formulas appearing in $Q * Y$. The only difficulty is that s', h' may fail to satisfy some formula of the form $v \mapsto w$ in $Q * Y$, because $h'(s'(v)) \neq s'(w)$. Suppose for contradiction this is the case.

Since $\text{dom}(h') = \text{dom}(h)$, we must have $s'(v) \in \text{dom}(h)$, and since $s', h \models A * X$, it must be that $s'(v)$ is covered by some formula in $A * X$. If there is a formula of the form $t \mapsto u$ in $A * X$ such that $s'(t) = s'(v)$, then we have $s'(w) = h(s'(v)) = h(s'(t)) = s'(u)$, since $s', h \models A * X$ and $s', h \models Q * Y$. But then $h'(s'(v)) = h'(s'(t)) = s'(u) = s'(w)$, since $s', h' \models t \mapsto u$, a contradiction. Therefore, there must be a formula $\text{array}(a, b)$ in $A * X$ such that $s'(a) \leq s'(v) \leq s'(b)$. But then, due to our initial choice of h , we know that $h(s'(v)) \neq s(w)$. Since the existential variables \mathbf{z} are not allowed to include w , this means $h(s'(v)) \neq s'(w)$, contradicting the fact that $s', h \models Q * Y$ (since it does not satisfy $v \mapsto w$). This completes the proof. \square

Lemma 5.20. *Given an instance G of the 2-round 3-colouring problem, the following statements are pairwise equivalent:*

- (a) *The biabduction problem (A_G, B_G) has a solution.*
- (b) *There is a winning strategy for the perfect colouring G .*
- (c) *$A_G \models B_G$ is valid.*

where A_G and B_G are the symbolic heaps given by Definition 5.19

Proof. We establish each direction of the above equivalences in turn.

(c) \Rightarrow (a)

This direction is trivial by taking

$$X = Y = \text{emp}$$

(b) \Rightarrow (c)

Suppose that there is a winning strategy such that every 3-colouring of the leaves can be extended to a perfect 3-colouring of the whole G . We will prove that $A_G \models B_G$. Let s, h be a stack-heap pair satisfying $s, h \models A_G$. The spatial part of A_G yields a decomposition of h as

$$h = \bigcirc_{i=1}^k h_i \circ \bigcirc_{(v_i, v_j) \in E} \widetilde{h_{ij}^{(e)}} \quad (1)$$

where for each $1 \leq i \leq k$, the h_i is a one-cell array, and for some b_i ,

$$\text{dom}(h_i) = \{s(d_i)\}, \text{ and } h_i(s(d_i)) = b_i \quad (2)$$

and, for each $(v_i, v_j) \in E$,

$$\text{dom}(\widetilde{h_{ij}^{(e)}}) = \{s(e_{ij}) + 1, s(e_{ij}) + 2, s(e_{ij}) + 3\} \quad (3)$$

Take the 3-colouring of the leaves obtained by assigning the colours $b_{i,1}$ to the leaves v_1, v_2, \dots, v_k resp.. where $1 \leq b_{i,1} \leq 3$, and $b_{i,1} - 1 \equiv b_i \pmod{3}$.

According to the winning strategy, we can assign colours, denote them by $b_{i,1}$, $i > k$, to the rest of vertices v_{k+1}, \dots, v_n , resp., obtaining a 3-colouring of the whole G such that no adjacent vertices share the same colour.

In addition, we mark edges (v_i, v_j) by $\widetilde{b_{ij}}$ complementary to $b_{i,1}$ and $b_{j,1}$.

We extend the stack s for quantified variables in B_G so that for all $i \leq k$,

$$s(c_{i,1}) = b_{i,1} = h_i(s(d_i)),$$

and, for each $(v_i, v_j) \in E$,

$$s(\widetilde{c_{ij}}) = 6 - b_{i,1} - b_{j,1}.$$

The fact that no adjacent vertices v_i and v_j share the same colour provides that

$$(s(c_{i,1}), s(c_{j,1}), s(\widetilde{c_{ij}}))$$

is a permutation of

$$(1, 2, 3),$$

resulting in that $s, \widetilde{h_{ij}^{(e)}}$ from (3) is also a model for

$$\text{array}(e_{ij}, c_{i,1}, c_{i,1}) * \text{array}(e_{ij}, c_{j,1}, c_{j,1}) * \text{array}(e_{ij}, \widetilde{c_{ij}}, \widetilde{c_{ij}})$$

Bringing all together, we get that s, h satisfies $s, h \models B_G$, which completes the proof of this direction.

(a) \Rightarrow (b)

Let $A_G * X \models B_G * Y$ and $A_G * X$ be satisfiable.

Since $A_G * X$ is satisfiable, there is a model of the form $s, h_A \circ h_X$ such that

$$s, h_A \models A_G, \text{ and } s, h_X \models X,$$

and, in particular,

$$h_A = \bigcirc_{i=1}^k h_i \circ \bigcirc_{(v_i, v_j) \in E} \widetilde{h_{ij}^{(e)}},$$

where for each $1 \leq i \leq k$, the h_i is a one-cell array such that

$$\text{dom}(h_i) = \{s(d_i)\}, \text{ and } h_i(s(d_i)) = s(c_i) \quad (4)$$

and, for each $(v_i, v_j) \in E$,

$$\text{dom}(\widetilde{h_{ij}^{(e)}}) = \{s(e_{ij}) + 1, s(e_{ij}) + 2, s(e_{ij}) + 3\} \quad (5)$$

We will construct the required winning strategy in the following way.

Assume a 3-colouring of the leaves be given by assigning colours, say $b_{i,1}$, to the leaves v_1, v_2, \dots, v_k respectively.

Then we modify our original stack s to a stack s' by setting, for each $1 \leq i \leq k$,

$$s'(c_i) = b_{i,1}.$$

with modifying thereby h_A to h'_A by means of replacing each h_i with the updated h'_i in which

$$h'_i(s(d_i)) = s'(c_i) = b_{i,1}.$$

We claim that still

$$s', h'_A \models A_G, \text{ and } s', h_X \models X,$$

and, therefore,

$$s', h'_A \circ h_X \models A_G * X.$$

The crucial point is that

- (a) First, c_i is quantified so that X cannot refer to c_i explicitly.
- (b) The only indirect possibility for X to refer to c_i by applying h_X to d_i is blocked by the fact that d_i is not in the domain of h_X .

Since $A_G * X \models B_G * Y$, we get

$$s', h'_A \circ h_X \models B_G * Y,$$

and for some $h_B \subseteq h'_A \circ h_X$ and stack s_B , which is extension of s' to the existentially quantified variables in B ,

$$s_B, h_B \models B.$$

Recall that B_G has been defined as follows:

$$\begin{aligned} \exists \mathbf{z}. & \left(\bigwedge_{i=1}^n (1 \leq c_{i,1} \leq 3) \wedge \bigwedge_{(v_i, v_j) \in E} (1 \leq \widetilde{c_{ij}} \leq 3) \right. \\ & \wedge \bigwedge_{i=1}^k (c_{i,1} - 1 \equiv c_i \pmod{3}) : \\ & *_{i=1}^k d_i \mapsto c_i * *_{(v_i, v_j) \in E} \text{array}(e_{ij}, c_{i,1}, c_{i,1}) \\ & * *_{(v_i, v_j) \in E} \text{array}(e_{ij}, c_{j,1}, c_{j,1}) * \text{array}(e_{ij}, \widetilde{c_{ij}}, \widetilde{c_{ij}}) \bigg). \end{aligned}$$

where the existentially quantified variables \mathbf{z} are all variables occurring in B_G that are not mentioned explicitly in A_G .

Because of $d_i \mapsto c_i$, we have for each $1 \leq i \leq k$,

$$s_B(c_i) = h'_i(s(d_i)) = s'(c_i) = b_{i,1},$$

which means that, for $1 \leq i \leq k$, these $s_B(c_i)$ represent correctly the original 3-colouring of the leaves.

Take the 3-colouring of the whole G obtained by assigning the colours $s_B(c_{i,1})$ to the rest of vertices v_{k+1}, \dots, v_n respectively.

The part of the form

$$\text{array}(e_{ij}, c_{i,1}, c_{i,1}) * \text{array}(e_{ij}, c_{j,1}, c_{j,1}) * \text{array}(e_{ij}, \widetilde{c_{ij}}, \widetilde{c_{ij}}),$$

provides that $s_B(c_{i,1}) \neq s_B(c_{j,1})$, which results in that no adjacent vertices v_i and v_j share the same colours $s_B(c_{i,1})$ and $s_B(c_{j,1})$, with providing a perfect 3-colouring of G .

This completes the direction, and the proof. \square

D. Proofs of results in Section 6

In order to prove Lemma 6.4, we make use of the following simple auxiliary lemma about the formula $\phi(-, -)$ from Definition 6.1.

Lemma D.1. *Let A and B be symbolic heaps with respective spatial parts:*

$$\begin{aligned} A : & *_{i=1}^n \text{array}(a_i, b_i) * *_{i=1}^k t_i \mapsto u_i \\ B : & *_{j=1}^m \text{array}(c_j, d_j) * *_{j=1}^{\ell} v_j \mapsto w_j \end{aligned}$$

Then we have, for any stack s ,

$$\begin{aligned} s \models \phi(A, B) \\ \Leftrightarrow \exists y \in \bigcup_{i=1}^n \{s(a_i), \dots, s(b_i)\} \cup \bigcup_{i=1}^k \{s(t_i)\} \text{ such that} \\ y \notin \bigcup_{j=1}^m \{s(c_j), \dots, s(d_j)\} \cup \bigcup_{j=1}^{\ell} \{s(v_j)\} \}. \end{aligned}$$

where $\phi(-, -)$ is given by Defn. 6.1.

Proof. Follows straightforwardly from the definitions of $\phi(-, -)$ and $\lfloor - \rfloor$. \square

Lemma 6.4. *For any instance (A, B) of the ASL entailment problem, and for any stack s ,*

$$s \models \chi(A, B) \Leftrightarrow \exists h. s, h \models A \text{ and } s, h \not\models B.$$

Proof. We assume that A and B are of the form given by Lemma D.1, and establish each direction of the lemma separately.

(\Rightarrow) Supposing that $s \models \chi(A, B)$, we require to construct a heap h such that $s, h \models A$ and $s, h \not\models B$. Note that $s \models \gamma(A)$ by assumption, so by Lemma 4.5 there is a heap h such that $s, h \models A$. Moreover, this fact is clearly independent of the data values stored in the arrays in A . Thus we may choose h such that $h(x) \neq s(w_j)$ for all $j \in [1, \ell]$ and for all $x \in [s(a_i), s(b_i)]$, where $i \in [1, n]$.

Now suppose for contradiction that $s, h \models B$. Thus, for some $\mathbf{q} \in \text{Val}^{|\mathbf{z}|}$ we have $s[\mathbf{z} \mapsto \mathbf{q}], h \models \text{qf}(B)$ (where \mathbf{z} is the tuple of existentially quantified variables in B). For convenience, we write $s' =_{\text{def}} s[\mathbf{z} \mapsto \mathbf{q}]$. Since \mathbf{z} does not include any variable in A , we also have $s', h \models A$. Thus

$$\text{dom}(h) = \bigcup_{i=1}^n \{s'(a_i), \dots, s'(b_i)\} \cup \bigcup_{i=1}^k \{s'(t_i)\}$$

and $h(s'(t_i)) = s'(u_i)$ for all $i \in [1, k]$. Similarly, since $s', h \models \text{qf}(B)$, we have

$$\text{dom}(h) = \bigcup_{j=1}^m \{s'(c_j), \dots, s'(d_j)\} \cup \bigcup_{j=1}^{\ell} \{s'(v_j)\}$$

and $h(s'(v_j)) = s'(w_j)$ for all $j \in [1, \ell]$. We note that, because of our restrictions on existential quantification, $s'(w_j) = s(w_j)$ for all w_j .

Now, since $s \models \chi(A, B)$, by instantiating the universal quantifiers $\forall \mathbf{z}$ in the second conjunct as \mathbf{q} , we obtain

$$s' \models \neg\gamma(\text{qf}(B)) \vee \phi(A, B) \vee \phi(B, A) \vee \psi_1(A, B) \vee \psi_2(A, B).$$

However, since $s', h \models \text{qf}(B)$, we have $s' \models \gamma(\text{qf}(B))$ by Lemma 4.5, and therefore

$$s' \models \phi(A, B) \vee \phi(B, A) \vee \psi_1(A, B) \vee \psi_2(A, B).$$

This gives us four disjunctive subcases to consider.

Case $s' \models \phi(A, B)$: In this case, Lemma D.1 and the two equations above for $\text{dom}(h)$ imply that there exists $y \in \text{dom}(h)$ such that $y \notin \text{dom}(h)$; contradiction.

Case $s' \models \phi(B, A)$: Symmetric to the case above.

Case $s' \models \psi_1(A, B)$: We have $s'(a_i) \leq s'(v_j) \leq s'(b_i)$ for some $i \in [1, n]$ and $j \in [1, \ell]$. On the one hand, we have $h(s'(v_j)) = s'(w_j) = s(w_j)$. On the other hand, h was chosen specifically such that $h(x) \neq s(w_j)$ for any $x \in [s(a_i), s(b_i)] (= [s'(a_i), s'(b_i)])$. Hence we have a contradiction.

Case $s' \models \psi_2(A, B)$: We have $s'(t_i) = s'(v_j)$ and $s'(u_i) \neq s'(w_j)$ for some $i \in [1, k]$ and $j \in [1, \ell]$. On the one hand we have $h(s'(t_i)) = s'(u_i) \neq s'(w_j)$, and on the other we

have $h(s'(t_i)) = h(s'(v_j)) = s'(w_j)$, a contradiction. This completes all subcases.

(\Leftarrow) Supposing that $s, h \models A$ but $s, h \not\models B$, we need to show that $s \models \chi(A, B)$. Since $s, h \models A$, we immediately get $s \models \gamma(A)$ by Lemma 4.5. Then, letting $\mathbf{q} \in \text{Val}^{|\mathbf{z}|}$ be an arbitrary instantiation of the variables \mathbf{z} and writing $s' = s[\mathbf{z} \mapsto \mathbf{q}]$, it remains show that

$$s' \models \neg\gamma(\text{qf}(B)) \vee \phi(A, B) \vee \phi(B, A) \vee \psi_1(A, B) \vee \psi_2(A, B).$$

Since \mathbf{z} does not mention any variable in A , we have $s', h \models A$, and thus

$$\text{dom}(h) = \bigcup_{i=1}^n \{s'(a_i), \dots, s'(b_i)\} \cup \bigcup_{i=1}^k \{s'(t_i)\}$$

with $h(s'(t_i)) = s'(u_i)$ for all $i \in [1, k]$.

Now, since $s, h \not\models B$, we can instantiating the quantifiers \mathbf{z} in B by \mathbf{q} to obtain $s', h \not\models \text{qf}(B)$. If $s', h \not\models \Pi'$, then immediately $s' \models \neg\gamma(\text{qf}(B))$ and we are done. Otherwise, s', h fails to satisfy the spatial part of $\text{qf}(B)$. By examining the satisfaction relation for spatial formulas, this yields four disjunctive subcases.

1. Some array in B is ill-defined under s' , i.e. $s'(c_j) > s'(d_j)$ for some j . In that case $s' \models \neg\gamma(\text{qf}(B))$, and we are done.
2. Each array in B is defined under s' , but $\text{dom}(h)$ is not, because the domains of the arrays and pointers in B overlap on some location. In this case, it is again straightforward to see that $s' \models \neg\gamma(\text{qf}(B))$.
3. The domain $\text{dom}(h)$ is well-defined, but not equal to

$$\bigcup_{j=1}^m \{s'(c_j), \dots, s'(d_j)\} \cup \bigcup_{j=1}^{\ell} \{s'(v_j)\}.$$

In that case, using Lemma D.1 and the characterisation of $\text{dom}(h)$ in terms of A above, it is easy to show that either $s' \models \phi(A, B)$ or $s' \models \phi(B, A)$.

4. Finally, it might be that $\text{dom}(h)$ agrees with the spatial part of B under s' (i.e. $s', h \models \lfloor \text{qf}(B) \rfloor$), but disagrees on some pointer value, i.e., $h(s'(v_j)) \neq s'(w_j)$ for some $j \in [1, \ell]$. We observe that $s'(v_j) \in \text{dom}(h)$, and distinguish two further subcases, using the previous characterisation of $\text{dom}(h)$ in terms of A above.

- If $s'(v_j) \in \{s'(a_i), \dots, s'(b_i)\}$ for some $i \in [1, n]$, then we immediately have $s' \models \psi_1(A, B)$.
- Otherwise, $s'(v_j) = s'(t_i)$ for some $i \in [1, k]$. In that case, $h(s'(v_j)) = h(s'(t_i)) = s'(u_i)$, and thus $s'(u_i) \neq s'(w_j)$. Thus $s' \models \psi_2(A, B)$, and we are done. This completes all subcases, and the proof. \square

Lemma 6.2. We can rewrite $\Phi(A, B)$ as a quantifier-free formula at only polynomial cost.

Proof. We write $\Phi(A, B) = \exists x. \alpha_{A,B}(x)$, so that, following Definition 6.1, $\alpha_{A,B}(x)$ is the formula

$$\bigvee_{i=1}^n a_i \leq x \leq b_i \wedge \bigwedge_{j=1}^m (x < c_j) \vee (x > d_j) .$$

We claim that $\Phi(A, B)$ is then equivalent to the formula

$$\bigvee_{i_0=1}^n \alpha_{A,B}(a_{i_0}) \vee \bigvee_{j_0=1}^m \alpha_{A,B}(d_{j_0} + 1) .$$

One direction of the equivalence is trivial (any stack satisfying the above formula immediately satisfies $\alpha_{A,B}(x)$ for some x and therefore $\Phi(A, B)$). We show the non-trivial direction.

Assuming that $s \models \Phi(A, B)$, there exists a number x_0 and $k \in [1, n]$ such that

$$s \models a_k \leq x_0 \leq b_k \wedge \bigwedge_{j=1}^m (x_0 < c_j) \vee (x_0 > d_j) .$$

We consider two cases, recalling that $\phi(A, B)$ captures the property that there is an address in an array in A that is not covered by any of the arrays in B (cf. Lemma D.1).

1. Suppose that the address $s(a_k)$ is not covered by any array in B , i.e., that $s(a_k) < s(c_j)$ or $s(a_k) > s(d_j)$ for all $j \in [1, m]$. In that case, trivially, $s \models \alpha_{A,B}(a_k)$, and we are done.
2. Otherwise, $s(a_k)$ is covered by an array in B , i.e., $s(c_j) \leq s(a_k) \leq s(d_j)$ for some $j \in [1, m]$. Then we choose d_{j_0} such that

$$s(d_{j_0}) = \max_{1 \leq j \leq m} \{s(d_j) \mid s(d_j) < x_0\} .$$

(That is, d_{j_0} is the largest right-endpoint of an array in B that is still smaller than x_0 .) In that case, the effect is that $s(d_{j_0} + 1)$ must still be covered by the arrays in A ,

$$s \models (a_k \leq d_{j_0} < d_{j_0} + 1 \leq x_0 \leq b_k)$$

but $s(d_{j_0} + 1)$ cannot itself be allocated in B

$$s \models \bigwedge_{j=1}^m ((d_{j_0} + 1 \leq x_0 < c_j) \vee ((x_0 > d_j) \wedge (d_{j_0} + 1 > d_j)))$$

Hence $\alpha_{A,B}(d_{j_0} + 1)$ holds, and we are done. \square

Lemma 6.7. *Let G be a 2-round 3-colouring instance, and let A_G and B_G be the symbolic heaps given by Defn. 6.6. Then, we have*

$$A_G \models B_G \Leftrightarrow \exists \text{ winning strategy for colouring } G.$$

Proof. Similar to Lemma 5.20. \square